Aufgabe 1: Schaltwerksentwicklung – Tintenstrahldrucker / Teil 2

a) Befüllen Sie die untenstehende Wahrheitstabelle für Übergangs- und Ausgangsfunktion des Schaltwerks aus Aufgabe 8 von *Übung 4: Schaltwerke*.

Berücksichtigen Sie beim Ausfüllen der Tabelle, dass diese als Basis für die Realisierung der Schaltung mit einem ROM dienen soll und dass *don't cares* entsprechend aufgelöst werden müssen! Geben Sie die Zuordnung der Signalleitungen zu den Datenleitungen des ROM an. Verwenden Sie für die Zustände nebenstehende dichte Zustandscodierung.

Zustand	$Z_1 Z_0$
Warten	00
Papiereinzug	01
Drucken	10
Fehler	11

Zustand		Eingänge			Übergang		Ausgänge		
Z_1	Z_0	A	P	T	D_1	D_0	E	D	F
A_0	A_1	A_2	A_3	A_4					
0	0	0	0	0					
0	0	0	0	1					
0	0	0	1	0					
0	0	0	1	1					
0	0	1	0	0					
0	0	1	0	1					
0	0	1	1	0					
0	0	1	1	1					
0	1	0	0	0					
0	1	0	0	1					
0	1	0	1	0					
0	1	0	1	1					
0	1	1	0	0					
0	1	1	0	1					
0	1	1	1	0					
0	1	1	1	1					
1	0	0	0	0					
1	0	0	0	1					
1	0	0	1	0					
1	0	0	1	1					
1	0	1	0	0					
1	0	1	0	1					
1	0	1	1	0					
1	0	1	1	1					
1	1	0	0	0					
1	1	0	0	1					
1	1	0	1	0					
1	1	0	1	1					
1	1	1	0	0					
1	1	1	0	1					
1	1	1	1	0					
1	1	1	1	1					

b) Zeichnen Sie die Gesamtschaltung des Schaltwerks! Übergangs- und Ausgangsfunktion sollen mit einem ROM-Baustein realisiert werden. Für die Zustandsspeicherung und Taktung (ein Wechsel zwischen Zuständen soll nur zum Takt möglich sein) sind Flip-Flops zu verwenden.
Vergessen Sie nicht, sämtliche Signale bzw. Leitungen Ihrer Schaltung entsprechend zu beschriften!

Aufgabe 2: Micro16 Architektur – Wahr oder falsch?

Welche Aussagen treffen zu? Begründen Sie Ihre Antwort!

(1)	Bei der Micro16 Architektur sind Datenwörter 8 Bit lang.	☐ richtig ☐ falsch
(2)	Bei der Micro16 Architektur werden Mikroinstruktionen durch eine 16 Bit Adresse adressiert.	☐ richtig ☐ falsch
(3)	Das *Memory Adress Register* (MAR) ist mit dem Adressbus verbunden.	☐ richtig ☐ falsch
(4)	Liegt an der *Memory Select* (MS) Leitung logisch 0 an, wird in das *Memory Buffer Register* (MBR) geschrieben, ansonsten wird daraus geladen.	☐ richtig ☐ falsch
(5)	Micro-Instruktionen sind beim Micro16 genau 16 Bit lang.	☐ richtig ☐ falsch
(6)	Die *Micro Sequencing Logic* kann Sprünge ausführen, indem sie den Folgewert des *Micro Instruction Counters* (MIC) bestimmt.	☐ richtig ☐ falsch
(7)	Am Ausgang Z der ALU liegt genau dann logisch 1 an, wenn im Ergebnis alle Bit logisch 0 sind.	☐ richtig ☐ falsch
(8)	Mithilfe des S-Bus Decoders werden Registerwerte über den A-Bus transferiert.	☐ richtig ☐ falsch
(9)	Die ALU-Operation Negation invertiert ein Datenwort bitweise.	☐ richtig ☐ falsch
(10)	Die Operation *right shift* entspricht einer Multiplikation des Datenworts mit der Zahl 2.	☐ richtig ☐ falsch
(11)	Die Control Unit teilt den Takt in drei Phasen und steuert damit die Ausführung der Mikroinstruktionen.	☐ richtig ☐ falsch

Aufgabe 3: Micro16 – Korrektheit von Instruktionen

Analysieren Sie die nachfolgenden Micro16-Instruktionen. Sofern Labels verwendet werden, gehen Sie davon aus, dass diese in gleicher Form definiert wurden. Grundsätzlich sind sowohl die Notation vom Lehrbuch als auch jene vom Micro-Simulator gültig.
Kreuzen Sie korrekte Instruktionen an und begründen Sie Ihre Antwort!

(1) ❍ `R9 ← R9∨R8`

(2) ❍ `R8 ← lsh(R1∧R2)`

(3) ❍ `R5 ← R6+R12`

(4) ❍ `R9 ← 2`

(5) ❍ `R0 ← R1-R2`

(6) ❍ `rsh(1); if Z goto 5`

(7) ❍ `MAR ← MBR; rd`

(8) ❍ `R3 ← lsh((¬R2)+R3))`

(9) ❍ `goto .R4`

(10) ❍ `R1 ← (R2∧1); if Z goto nextStep`

(11) ❍ `MBR ← R1; R5 ← R0+R1`

(12) ❍ `MAR ← R1; MBR ← R0+R1; R2 ← R0+R1; wr`

Allgemeiner Hinweis: Speziell für diese Aufgabe und die nachfolgenden Implementierungsaufgaben steht Ihnen optional ein Micro16-Simulator im TUWEL-Kurs zur Verfügung. Details zur Handhabung entnehmen Sie bitte den Dokumenten im TUWEL-Abschnitt *Micro16*.
Für die Implementierungsaufgaben (Aufgaben 6 bis 8) gibt es eine zusätzliche TUWEL-Aktivität *Hochladen Micro16-Code*, in der bei Verwendung des Simulators bis zu zwei Zusatzpunkte erreicht werden können (für Details siehe Aktivität).
Sämtliche Aufgaben können aber auch ohne Verwendung des Simulators gelöst werden!

Aufgabe 4: Micro16 – Multiplikation

Entnehmen Sie Ihrer Matrikelnummer die beiden Zahlen x und y wie folgt: x entspricht der 3., 5. und 6. Stelle, y der 4. und 7. Stelle Ihrer Matrikelnummer.

Beispiel: Für die Matrikelnummer 1234567 ist somit $x = 356$ und $y = 47$.

a) Wandeln Sie die Zahlen x und y ins Binärsystem um!

b) Stellen Sie die beiden Zahlen in Zweierkomplementdarstellung dar! Verwenden Sie dabei die Datenwortlänge des Micro16.

c) Konstruieren Sie die beiden Zweierkomplement-Zahlen durch möglichst wenige Micro16-Instruktionen und legen Sie diese in Register R5 (x) bzw. R6 (y) ab! Sie können auf die Konstanten 0, +1, −1 direkt zugreifen. Geben Sie Ihre Instruktionen in symbolischer Notation an:

```
R5 ← lsh(1+1)   # lege (+4)10 in R5 ab
R5 ← lsh(R5+1)  # überschreibe R5 mit (+10)10
```

d) Führen Sie anschließend den nachfolgenden Programmcode mit Ihren Werten für R5 und R6 aus. Geben Sie die Inhalte der Register R5 und R6 immer zum Zeitpunkt von `loop:` (2. Programmzeile) in Binärdarstellung in untenstehender Tabelle an! Sobald das Programm terminiert, lassen Sie nachfolgende Zeilen leer.

Hinweis: Im TUWEL-Kurs steht der Programmcode als *Multiplikation_UE.txt* zur Verfügung.

```
     R7 ← 0                 #
loop:
     (R6∧1); if Z goto zero  #
     R7 ← R7+R5             #
zero:
     R6 ← rsh(R6)           #
     (R6); if Z goto end    #
     R5 ← lsh(R5)           #
     goto loop
end:
```

loop	Register R5	Register R6
1		
2		
3		
4		
5		
6		
7		

e) Überlegen Sie sich die Funktionsweise dieses Multiplikationsalgorithmus und ergänzen Sie entsprechende Kommentare in obigem Programmcode nach den #-Symbolen!

Aufgabe 5: Micro16 – Analyse von Hex-Code

Gegeben ist der folgende Micro16-Code in hexadezimaler Notation:

18160500 08161600 08166400 01A07600 00200000

a) Übersetzen Sie die Instruktionen in binären Micro16-Code!

AMUX	COND	ALU	SH	MBR	MAR	RD/WR	MS	ENS	S-BUS	B-BUS	A-BUS	ADR

b) Geben Sie die Instruktionen in symbolischer Notation an! Die Adressierung der Register und der Konstanten erfolgt in Anlehnung an die Architektur der Vorlesung wie folgt:

Register	Adresse
0	0000
+1	0001
-1	0010
R0	0100
R1	0101
...	...
R10	1110

c) Welche Funktion wird durch Ausführung dieser Mikroinstruktionen realisiert?
Hinweis: Gehen Sie davon aus, dass in R1 eine Zahl in Zweierkomplentdarstellung vorliegt.

Aufgabe 6: Micro16 – Implementierung – Funktion `mod5()`

Entwerfen Sie ein Micro16-Programm für die Funktion `mod5()`:
Diese Funktion liest das Datenwort an der Speicheradresse $(16)_{10}$ ein und berechnet die Restklasse zur Basis $(5)_{10}$ vom eingelesen Datenwort. Das Ergebnis soll in Register R1 abgelegt werden. Das eingelesene Datenwort ist dabei immer eine positive Binärzahl $< 2^{15}$.

Beispiel 1: Wert an Adresse $(16)_{10}$: 00000000 00011010
R1 nach Programmausführung: 00000000 00000001

Beispiel 2: Wert an der Adresse $(16)_{10}$: 00000000 01110111
R1 nach Programmausführung: 00000000 00000100

Aufgabe 7: Micro16 – Hamming-Distanz

Schreiben Sie ein Micro16-Programm zur Berechnung der Hamming-Distanz zwischen zwei binären Zeichenketten der Länge 16 Bit.
Die Hamming-Distanz ist ein Maß für die Unterschiedlichkeit von Zeichenketten. Der Wert der Hamming-Distanz entspricht dabei der Anzahl der unterschiedlichen Stellen.

Beispiel: $10\underline{1}1\underline{0}$ und $10\underline{1}0\underline{0} \rightarrow$ Hamming-Distanz = 1
$1\underline{0}11\underline{0}$ und $1\underline{1}111 \rightarrow$ Hamming-Distanz = 2

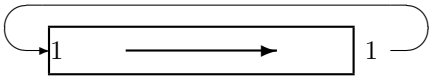
Register R8 beinhaltet die erste binäre Zeichenkette, Register R9 die zweite. Register R10 soll schließlich den errechneten Wert der Hamming-Distanz als Zweierkomplementzahl beinhalten.

Aufgabe 8: Micro16 – Nichtinvertierender Ringzähler

Entwerfen Sie ein Micro16-Programm für einen rechtslaufenden, nichtinvertierenden Ringzähler.

Bei dieser Form eines Schieberegisters wird der ursprüngliche Wert des niederwertigsten Bits (*lsb*) eines Registers auf das höchstwertige Bit (*msb*) desselben Registers übertragen, nachdem alle anderen Bits dieses Registers um eine Stelle nach rechts verschoben wurden.

Beispiel: Ist die Bitfolge '10000000 00000011', so lautet das Ergebnis nach einem Schritt '11000000 00000001'.



Die zu schiebende Bitfolge liegt in Register R0. Die Anzahl auszuführender Schiebeoperationen ist in Register R1 abgelegt. Das Endergebnis soll auf die Speicheradresse $(3FFF)_{16}$ geschrieben werden, wobei auch nach jedem Schiebeschritt der aktuelle Registerinhalt an dieser Adresse abgelegt werden soll.