VU Technische Grundlagen der Informatik **Übung 6: Mikroprozessor (Micro16)**
183.579, 2017W Übungsgruppen: Mo., 18.12. – Mi., 20.12.2017

Aufgabe 1: Schaltwerksentwicklung – Kaffeemaschine / Teil 2

a) Befüllen Sie die untenstehende Wahrheitstabelle für Übergangs- und Ausgangsfunktion des Schaltwerks aus Aufgabe 9, Übungsblatt 5.

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. Verwenden Sie für die Zustände nebenstehende dichte Zustandscodierung.

Zustand	Z_1Z_0
Standby	00
Heizen	10
Zubereiten	01
Fehler	11

Zustand		Eingänge			Übergang		Ausgänge		
Z_1	Z_0	F	S	T	D_1	D_0	H	L	V

b) Zeichnen Sie die Gesamtschaltung des Schaltwerks, wobei Sie für die Realisierung der Übergangs- und Ausgangsfunktion einen ROM verwenden sollen. Vergessen Sie nicht, die Adress- und Datenleitungen des ROMs den Signalen Ihrer Schaltung zuzuordnen.

Aufgabe 2: Micro16-Architektur – Wahr oder falsch?

Welche Aussagen treffen zu? Begründen Sie Ihre Antwort.

(1)	Das *Memory Buffer Register* (MBR) wird über den A-Bus beschrieben.	☐ richtig ☐ falsch
(2)	Bei einem schreibenden Zugriff auf den Speicher müssen für mindestens zwei Takte die Leitung read/write im Zustand logisch 0 und die MS-Leitung auf logisch 0 gehalten werden.	☐ richtig ☐ falsch
(3)	Die *Arithmetic Logic Unit* (ALU) beherrscht insgesamt vier arithmetische Operationen.	☐ richtig ☐ falsch
(4)	Die *Micro Sequencing Logic* kann Sprünge ausführen, indem sie den Folgewert des *Micro Instruction Counters* (MIC) anhand der Ausgänge des Shifters bestimmt.	☐ richtig ☐ falsch
(5)	Am Ausgang N der ALU liegt genau dann logisch 1 an, wenn alle Bits im Ergebnis logisch 1 sind.	☐ richtig ☐ falsch
(6)	Mithilfe des A-Bus Decoders werden Registerwerte über den A-Bus transferiert.	☐ richtig ☐ falsch
(7)	Der Shifter erlaubt das Ergebnis der ALU zu verdoppeln/halbieren.	☐ richtig ☐ falsch
(8)	Die Größe des Micro16-Programmspeichers beträgt 2^{16} Bit.	☐ richtig ☐ falsch
(9)	Die Control Unit teilt den Takt in vier Phasen und steuert damit die Ausführung der Mikroinstruktionen.	☐ richtig ☐ falsch
(10)	Das MAR (Memory Address Register) ist 16 Bit groß und besitzt 0xFFFF als höchste Adresse	☐ 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 Micro16-Simulator gültig.
Kreuzen Sie korrekte Instruktionen an und begründen Sie Ihre Antwort!

(1) ❍ `R4 ← R5+1`

(2) ❍ `(R10+R2); if C goto 33`

(3) ❍ `R10 ← rsh(R5)+R8`

(4) ❍ `R7 ← R7*R1`

(5) ❍ `R10 ← -1+1`

(6) ❍ `goto 712`

(7) ❍ `(rsh(¬0)); if N goto Zeile712`

(8) ❍ `R5 ← (¬R7+R0)`

(9) ❍ `R8 ← 33`

(10) ❍ `MBR ← +1; R2 ← (-1)+1; 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*.
Sämtliche Aufgaben können aber auch ohne Verwendung des Simulators gelöst werden.
Für die Implementierungsaufgaben Aufgaben 8 und 9 gibt es eine zusätzliche TUWEL-Aktivität *Hochladen Micro16-Code*, bei der bis zu zwei Zusatzpunkte erreicht werden können (für Details, insbesondere Dateibenennung innerhalb der hochzuladenden Zip-Datei, siehe Aktivität *Hochladen Micro16-Code*).

Aufgabe 4: Micro16 – Programm-Analyse

Gegeben ist der folgende Micro16-Programmcode inklusive initialer RAM-Belegung:

```
00:  R0  ← rsh(-1)
01:  MAR ← R0; rd
02:  rd
03:  R0  ← MBR
04:  R2  ← lsh(1+1)
05:  R2  ← lsh(R2)
06:  MAR ← R2; rd
07:  rd
08:  R2  ← MBR
09:  R1  ← 0
10:  R2  ← ¬R2
11:  R2  ← R2+1
12:  loop:
13:      (R0+R2); if N goto end
14:      R0 ← R0+R2
15:      R1 ← R1+1
16:      goto loop
17:  end:
```

RAM-Adresse	Wert
⋮	⋮
0x0005	0000 0000 0000 1101
0x0006	0000 0000 0000 1111
0x0007	0000 0000 0000 0101
0x0008	0000 0000 0000 0011
⋮	⋮
0x7FFE	0000 0000 0001 1101
0x7FFF	0000 0000 0000 1010
0x8001	0000 0000 0001 1111
0x8002	0000 0000 0001 0011
⋮	⋮
0xFFFE	0000 0000 0000 1111
0xFFFF	0000 0000 0001 1011

a) Wie lautet der Inhalt des `MBR` in den Codezeilen `03` und `08`?

b) Auf welche RAM-Adresse wird in Codezeilen `02` und `07` zugegriffen?

c) Tragen Sie die Registerinhalte von `R0` und `R1` direkt vor jeder Ausführung von `loop:` in die Tabelle ein. Sobald das Programm terminiert, lassen Sie nachfolgende Tabellenzeilen frei. Führende Nullen können weggelassen werden.

`loop`	Register `R0`	Register `R1`
0		
1		
2		
3		
4		

d) Welche mathematische Funktion realisiert dieses Programm?

Aufgabe 5: Micro16 – Codierung eines Microprogramms

Übersetzen Sie die nachfolgenden drei Micro16-Instruktionen A-E in Micro16-Code! Bei den Instruktionen handelt es sich um korrekte Micro16-Instruktionen.

Instruktion A: `R1 ← ~R1`

Instruktion B: `MBR ← ~MBR`

Instruktion C: `MBR ← R1 & MBR`

Instruktion D: `MAR ← (-1); wr`

Instruktion E: `wr`

a) Tragen Sie die Instruktionen in binärer Form in die nachfolgende Tabelle ein! 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

	AMUX	COND	ALU	SH	MBR	MAR	RDWR	MS	ENS	S-BUS	B-BUS	A-BUS	ADR
A:													
B:													
C:													
D:													
E:													

b) Auf welche Adresse wird in der Micro16-Instruktion D zugegriffen?

c) Welches Logikgatter wird durch das Micro16-Programm für die beiden Operanden R1 und MBR realisiert?

d) Könnten die beiden ursprünglichen Micro16-Instruktionen C und D zu einer einzigen neuen Micro16-Instruktion X zusammengefasst werden?

Instruktion X: `MAR ← (-1); MBR ← R1 & MBR; wr`

Begründung:

Aufgabe 6: Micro16 – Implementierung einer Bit-Sortierung

Schreiben Sie ein Micro16-Programm, das eine in Register R0 abgelegte Bitfolge wie folgt sortiert: Alle enthaltenen Einsen sollen rechts angeordnet werden, alle enthaltenen Nullen links.

Beispiel: R0 vor Programmausführung: 00100101 00001100
R0 nach Programmausführung: 00000000 00011111

Aufgabe 7: Micro16 – Primzahlentest

Schreiben Sie ein Micro16-Programm zur Berechnung von Primzahlen im Zahlenbereich 1-32 durch Probedivision. Die Primzahlen sollen beginnend ab Adresse 0 abgespeichert werden. Initial ist die Primzahl 2 in der Adresse 0 abgespeichert.

Hinweis: Bei der Probedivision probiert man nacheinander, ob die Zahl n durch eine der Primzahlen p ($2 \leq p \leq \sqrt{n}$) teilbar ist. Wenn nicht, dann ist n eine Primzahl. Zur Vereinfachung führen Sie die Probedivision für n im Zahlenbereich $2 \leq p \leq (n-1)$ durch.

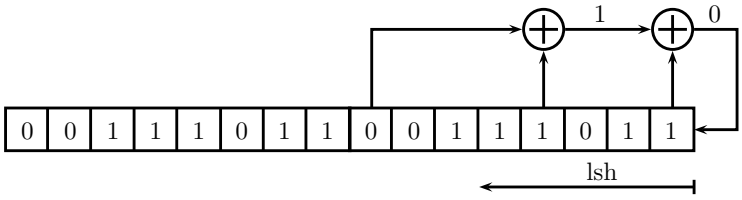
Könnte die Probedivision in einem kleineren einfach zu erstellenden Zahlenbereich durchgeführt werden (Begründung)?

Aufgabe 8: Micro16 – Zufallszahlengenerator nach Tausworthe

Einer der einfachsten Zufallszahlengeneratoren (nach Tausworthe) lässt sich mittels rückgekoppeltem Schieberegister realisieren. Hierbei ergibt sich das Bit, welches von einer Seite in das Schieberegister "hinein geschoben wird", durch XOR-Verknüpfung einzelner Bits des ursprünglichen Registerinhaltes. Der Registerinhalt entspricht der generierten (Pseudo-)Zufallszahl.

Schreiben Sie ein Micro16-Programm welches einen Tausworthe Zufallszahlengenerator realisiert. Das Programm soll den Startwert für die Zufallszahlengenerierung (*seed*) aus dem Speicher an der Adresse 0x7FFF auslesen und in das Register R0 speichern. Die erste Zufallszahl wird dann berechnet, indem das 1. Bit (LSB), das 4. Bit und das 8. Bit XOR-verknüpft werden. Das Resultat dieser Verknüpfung wird dann als neues LSB für Register R0 verwendet, nachdem es einem Linksshift unterzogen wurde. Die so erzeugte Zufallszahl soll danach wieder in den Speicher auf die Adresse 0x7FFF zurückgeschrieben werden.

Beispiel: 0x7FFF: 1000 0001 $\underline{0}$010 $\underline{0}$11$\underline{1}$ $\Rightarrow$ R0 $\leftarrow$ 0000 0010 0100 111$\underline{1}$
0x7FFF: 0000 0010 $\underline{0}$100 $\underline{1}$11$\underline{1}$ $\Rightarrow$ R0 $\leftarrow$ 0000 0100 1001 111$\underline{0}$



Aufgabe 9: Micro16 – Paritätsbit

Ein Paritätsbit kann zur Erkennung von Datenwortfehlern verwendet werden. Das Paritätsbit einer Folge von Bits dient als Ergänzungsbit, um die Anzahl der mit 1 belegten Bits (inklusive Paritätsbit) der Folge als gerade oder ungerade zu ergänzen. Man spricht von gerader Parität, wenn die Anzahl der mit 1 belegten Bits in der Folge gerade ist, andernfalls ungerader Parität.

Schreiben Sie ein Micro16-Programm für die Berechnung der geraden Parität eines Datenworts. Das Programm liest das Datenwort an der Speicheradresse $(8)_{10}$ ein. Das Paritätsbit soll am Ende des Programms im LSB in Register R1 abgelegt werden.

Beispiel 1:	Wert an Adresse $(8)_{10}$:	00000000 00011010
	Paritätsbit:	1
	R1 nach Programmausführung:	00000000 00000001
Beispiel 2:	Wert an der Adresse $(8)_{10}$:	00000000 01110111
	Paritätsbit:	0
	R1 nach Programmausführung:	00000000 00000000