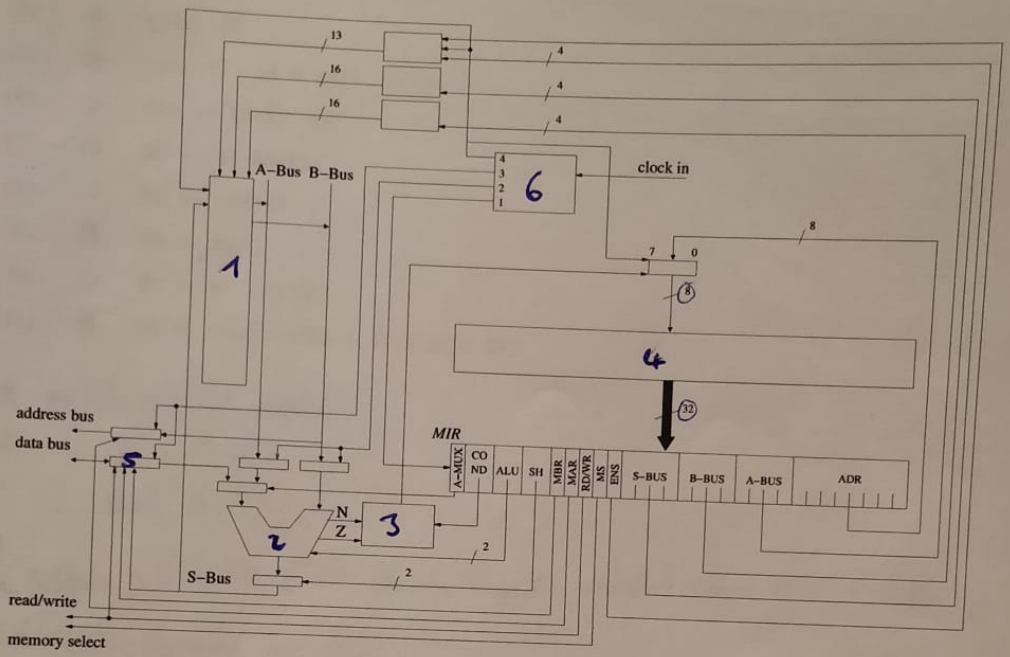
TG1 – UE6

VU Technische Grundlagen der Informatik
183.579, 2021W

Übung 6: Mikroprozessor (Micro16)
Übungsgruppen: Mo., 13.12. – Mi., 15.12.2021

Aufgabe 1: Micro16-Architektur – Aufbau und Funktionsweise

Nachfolgende Abbildung zeigt Ihnen den Aufbau des Micro16.



a) Fügen Sie in der Abbildung die entsprechenden Ziffern für die angegebenen Komponenten des Micro16 ein.

- Register Scratchpad/Register File: (1)
- Arithmetic Logic Unit (ALU): (2)
- Micro-Sequencing Logic: (3)
- Micro-Programmspeicher: (4)
- Memory-Buffer Register (MBR): (5)
- 4-Phase Clock (Control-Unit): (6)

b) Beschreiben Sie kurz die Funktionsweise der oben genannten Komponenten des Micro16.

c) Erklären Sie anhand der Abbildung die Größe des Micro-Programmspeichers.

c) 8 Bit Eingang → 256 ; 32 Bit Ausgang ; 256 × 32 Bit

b) 1) 16 Register, verbunde mit ALU
2) Parallele Addition, Bitweise UND-Verknüpfung, Bitweise Komplementbildung, Durchschalten eines Datenworts
3) verantwortlich für Sprünge 00 → nichts, 01 → Sprung bei N=1, 10 → Sprung bei Z=1, 11 → unbedingter Sprung
4) Speicher für Microprogramm Instruktionen
5) Schnittstelle für Datenaustausch, Laden von Registerinhalt
6) Trigger C1. Takt Laden Microinstruktion laden, Daten auf A- und B-Bus legen, Auswahl für ALU und Shift Register C2. Register A und B mit Daten laden, C3. ALU und Shifter Ziel für gewählt Operation geben C4. Ergebnis vom S-Bus in das Zielregister laden

Aufgabe 2: 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) ☒ `MAR ← R3; MBR ← 1+R3; R3 ← 1+R3; wr`

(2) ○ `R7 ← 1100`

(3) ○ `MBR ← R2; R0 ← R2+(-1)`

(4) ☒ `goto .R0`

(5) ☒ `rsh(-1); if N goto .loop`

(6) ○ `MAR ← R6+R0; rd`

(7) ○ `R9 ← R4-R5+R13`

(8) ○ `R6 ← ¬R6+R7`

(9) ☒ `R8 ← R1∧1`

(10) ○ `R3 ← R7∧rsh(R1)`

(11) ☒ `R2 ← rsh(-1∧MBR); if N goto 255`

1) MAR wird auf R3 gesetzt
MBR auf 1+R3
R3 auf 1+R3
~~falsch~~
Zum Schluss ein Write → da gleiche Register möglich sonst noch ein Bsp

4) ~~falsch~~ unbedingter Sprung zu R0

5) Wenn MSB von −1 = negativ → Sprung zu loop

9) ~~falsch~~ Verbinden von R1 und 1
Speichern in R8

11) Speichern von right shift(−1 und MBR in R2, wenn negativ → Sprung zu Zeile 255

c)

Allgemeiner Hinweis: Speziell für die nachfolgende Aufgabe und die nachfolgenden Implementierungsaufgaben steht Ihnen 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.

Aufgabe 3: Micro16 – Programm-Analyse

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

```
00:  R0  ← lsh(1+1)
01:  R1  ← rsh(-1)
02:  R1  ← rsh(R1)
03:  R1  ← ¬R1
04:  R1  ← R1+1
05:  R4  ← lsh(1+1)
06:  :loop
07:      (R4); if Z goto .end
08:      R4 ← R4+(-1)
09:      R0 ← R0+1
10:      MAR ← R0; rd
11:      rd
12:      R2 ← ¬R1
13:      R2 ← R2+1
14:      (MBR+R2); if N goto .loop
15:         R1 ← MBR
16:         goto .loop
17:  :end
```

RAM-Adresse	Wert
⋮	⋮
0x0005	1110 0000 0100 1001
0x0006	0000 0000 0000 1011
0x0007	1110 1101 1000 0000
0x0008	0000 0000 0010 0011
0x0009	1000 0000 0000 1111
⋮	⋮
0x7FFF	0000 0000 0000 0001
0x8000	0000 0000 0001 0000
0x8001	0000 0000 0001 0001
0x8002	0000 0000 0000 0001
0x8003	0000 0000 0010 0000
⋮	⋮
0xFFFF	0000 0000 0001 1110

a) Wie lautet der Inhalt des `MBR` in der Codezeile `11`? (Falls öfters ausgeführt bitte für jedes mal angeben.)

b) Auf welche RAM-Adresse wird in der Codezeile `10` zugegriffen?
(Falls öfters ausgeführt bitte für jede Ausführung angeben.)

c) Tragen Sie die Registerinhalte von `R1` und `R2` direkt nach jeder Ausführung der Codezeile `loop:` in die Tabelle ein. Sobald das Programm terminiert, lassen Sie nachfolgende Tabellenzeilen frei. Führende Nullen können weggelassen werden.

loop	Register R1	Register R2
0	0xC001	0x0000
1	0x0001	0x3FFF
2	0xED80	0x3FFF
3	0xED80	0x1280
4	0xED80	0x1280

d) Welche mathematische Funktion realisiert dieses Programm? Wird diese immer korrekt berechnet?
(Überlegen Sie sich für die zweite Frage auch, was passieren würde, wenn die Bedingung in Zeile 07 `if N goto .end` lauten würde, die „Schleife“ also ein weiteres mal durchlaufen werden würde.)

a)

loop	R1	R2
0	0xC001	0x0000
1	0xE049	0x3FFF
2	0x000D	0x1FD7
3	0x0003	0xFFF5
4	0x0023	0xFFF5

Terminierungsschritt

0xC001 = -16383
0xE049 = -8119
0x000D = 11

0x0023 = 35

Aufgabe 4: Micro16 – Codierung eines Microprogramms

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

Instruktion A: `AC ← ~AC`

Instruktion B: `AC ← MBR & AC`

Instruktion C: `MAR ← (-1); MBR <- ~AC; wr`

Instruktion D: `wr`

Instruktion E: `R0 ← rsh(R0) goto 255`

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:

Codierung	Beschreibung
ALU = 00	A-Bus unverändert durchschalten
ALU = 01	Addition (linker Operand auf A-Bus)
ALU = 10	bitweises UND (linker Operand auf A-Bus)
ALU = 11	bitweise Negation (A-Bus)
SH = 01	shift left
SH = 10	shift right
COND = 01	Sprung auf N=1
COND = 10	Sprung auf Z=1
COND = 11	unbedingter Sprung

Register	Adresse
0	0000
+1	0001
-1	0010
PC	0011
R0	0100
R1	0101
...	...
R10	1110
AC	1111

Sprung — ob S-Bus in MBR / ob B in MAR (speichern wir)

ob MBR als A-Eingang der ALU verwendet wird

	AMUX	COND	ALU	SH	MBR	MAR	RDWR	MS	ENS	S-BUS	B-BUS	A-BUS	ADR
A:	0	00	11	00	0	0	0	0	1	1111	0000	1111	0000 0000
B:	1	00	10	00	0	0	0	0	1	1111	1111	0000	0000 0000
C:	0	00	11	00	1	1	0	1	0	0000	0010	1111	0000 0000
D:	0	00	00	00	0	0	0	1	0	0000	0000	0000	0000 0000
E:	0	11	00	10	0	0	0	0	1	0100	0000	0100	1111 1111

0 wr / 1 rd ← MS=1

↑ Sprungziel für COND

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

0xFFFF

c) Welche Boole'sche Funktion wird durch das Micro16-Programm für die beiden Operanden `AC` und `MBR` realisiert?

MBR → AC

MBR	AC	
0	0	1
1	0	0
0	1	1
1	1	1

Aufgabe 5: Micro16 – Decodierung eines Microprogramms

Gegeben sind nachfolgende Bitfolgen A–E, die der Reihe nach in das Microinstruction Register (MIR) des Micro16 geladen werden.

	AMUX	COND	ALU	SH	MBR	MAR	RDWR	MS	ENS	S-BUS	B-BUS	A-BUS	ADR
A:	0	00	11	00	0	0	0	0	1	1001	0000	1001	0000 0000
B:	1	00	11	00	1	0	0	0	0	0000	0000	0000	0000 0000
C:	1	00	10	00	0	0	0	0	1	1100	1001	0000	0000 0000
D:	0	00	11	00	1	1	0	1	0	0000	0001	1100	0000 0000
E:	0	00	00	00	0	0	0	1	0	0000	0000	0000	0000 0000

a) Decodieren Sie die Bitfolgen in Micro16-Code. Bei den Bitfolgen handelt es sich um korrekte Micro16-Instruktionen.

Codierung	Beschreibung
ALU = 00	A-Bus unverändert durchschalten
ALU = 01	Addition (linker Operand auf A-Bus)
ALU = 10	bitweises UND (linker Operand auf A-Bus)
ALU = 11	bitweise Negation (A-Bus)
SH = 01	shift left
SH = 10	shift right
COND = 01	Sprung auf N=1
COND = 10	Sprung auf Z=1
COND = 11	unbedingter Sprung

Register	Adresse
0	0000
+1	0001
-1	0010
PC	0011
R0	0100
R1	0101
...	...
R10	1110

Instruktion A: R5 ← ~R5

Instruktion B: MBR ← ~MBR

Instruktion C: R8 ← MBR ∧ R5

Instruktion D: MAR ← 1; MBR ← ~R8; wr

Instruktion E: wr

b) Interpretieren Sie den Code bestehend aus den Instruktionen A–E. Welche Boole'sche Funktion wird hier realisiert? Was sind die Operanden und wo steht nach Ausführung von Instruktion E schlussendlich das Ergebnis?

R5 ∨ MBR bei der Adresse 0x0001

R5	MBR	
0	0	0
1	0	1
0	1	1
1	1	1

Für die Implementierungsaufgaben 6-9 gibt es eine zusätzliche TUWEL-Aktivität *Hochladen Micro16-Code*. Bitte laden Sie dort Ihren Mirco16 Code hoch (für Details, insbesondere Dateibenennung innerhalb der hochzuladenden Zip-Datei, siehe Aktivität *Hochladen Micro16-Code*).

Aufgabe 6: Micro16 – GGT Berechnung

Schreiben Sie ein Micro16-Programm zur Berechnung des größten gemeinsamen Teilers von R0 und R1. Dabei können Sie davon ausgehen, dass R0 $\geq$ R1 und R1, R0 ≥ 0 gilt. Der größte gemeinsame Teiler soll nach Programmausführung im Register R2 stehen. *Hinweis:* Sie können für die Berechnung den euklidischen Algorithmus verwenden.

Beispiel: R0 vor Programmausführung: $(15)_{10}$
R1 vor Programmausführung: $(12)_{10}$
R2 nach Programmausführung: $(3)_{10}$

R1 = 12

R1 ← 1+1 | 10
R1 ← R1+1 | 11
R1 ← lsl(R1) | 110
R1 ← lsl(R1) | 1100

R0 = 15

R0 ← 1+1 | 10
R0 ← R0+1 | 11
R0 ← lsl(R0) | 110
R0 ← R0+1 | 111
R0 ← lsl(R0) | 1110
R0 ← R0+1 | 1111

Bsp Input

: start
(-R1) in R2
R2 ← ~R1
R2 ← R2+1

R0 - R1
R0 ← R0+R2

Wenn Rest = 0 dann zu .end
R0; if Z goto .end
Wenn Rest < 0 dann zu .zeh variable
R0; if N goto .zeh variable
goto .start

: zeh Variable
R2 ← R1
R1 ← R0+R1
R0 ← R2
Wenn R1 (Rest) = 0
R1; if Z goto .end
goto .start

: end
R2 ← R1

R0	R1	R2
15 = 1 · 12 + 3		
12 = 4 · 3 + 0		

Aufgabe 7: 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 auf eine gerade oder ungerade Zahl zu ergänzen. Man spricht von gerader Parität, wenn die Anzahl der mit 1 belegten Bits in der Folge gerade ist, andernfalls von ungerader Parität.

Schreiben Sie ein Micro16-Programm für die Berechnung der ungeraden Parität eines Datenworts. Das Programm liest das Datenwort an der Speicheradresse $(FFFF)_{16}$ ein. Das Paritätsbit soll am Ende des Programms im LSB in Register R1 abgelegt werden.

Beispiel 1: Wert an Adresse $(FFFF)_{16}$: 00000000 00011010
Paritätsbit: 0
R1 nach Programmausführung: 00000000 00000000

Beispiel 2: Wert an der Adresse $(FFFF)_{16}$: 00000000 01110111
Paritätsbit: 1
R1 nach Programmausführung: 00000000 00000001

ungerade 1 ⇒ 0
gerade 1 ⇒ 1
Counter; wenn LSB = 1 ⇒ ungerade 1

```
#Einlesen an 0xFFFF
MAR ← (-1); rd
rd
R0 ← MBR

:start
R0; if z goto .evaluation

#letzte Bit entfernen
R1 ← rsh(R0)
R1 ← lsh(R1)

#R0-R1 ⇒ LSB R0
R1 ← ~R1
R1 ← R1+1
R1 ← R0+R1

#letzte Stelle entfernen
R0 ← rsh(R0)
#LSB = 0
R1; if z goto .start
#LSB = 1
R2 ← R2 + 1

goto .start

:evaluation
R0 ← R2
#letzte Bit entfernen
R1 ← rsh(R0)
R1 ← lsh(R1)

# R0-R1 ⇒ LSB-R0
R1 ← ~R1
R1 ← R1+1
R1 ← R0+R1

R1; if z goto .von0zu1
goto .von1zu0

:von0zu1
R1 ← 1
goto .end

:von1zu0
R1 ← 0
goto .end

:end
```

Aufgabe 8: 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: 1011$\underline{0}$ und 1010$\underline{0}$ → Hamming-Distanz = 1
$\underline{1}$011$\underline{0}$ und $\underline{1}$111$\underline{1}$ → 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.

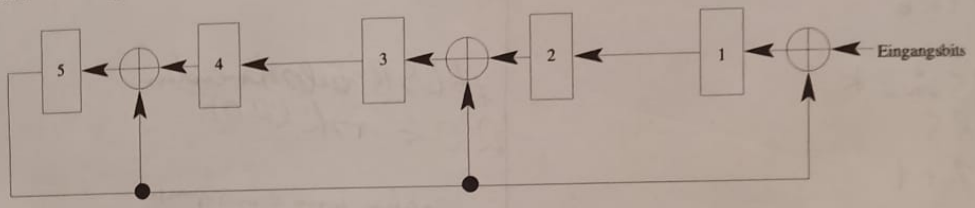
```
:loop
#Get first Digit Zeichenkette 1
R0 ← rsh(R8)
R0 ← lsh(R0)
R0 ← ~R0
R0 ← R0+1
R0 ← R8+R0

# Get first Digit Zeichenkette 2
R1 ← rsh(R9)
R1 ← lsh(R1)
R1 ← ~R1
R1 ← R1+1
R1 ← R9+R1

#Check if Digits are not equal
R5 ← ~R1
R5 ← R5+1
(R5+R0); if Z goto :continue
R6 ← ~R0
R6 ← R6+1
(R6+R1); if Z goto :continue
goto .addCounter
:continue

#Beide LS Bit entfernen
R8 ← rsh(R8)
R9 ← rsh(R9)

# Reached last Digit?
R8; if Z goto .end
goto .loop

:addCounter
R10 ← R10+1
goto .continue

:end
#Hamming Distanz in 2K
R10 ← ~R10
R10 ← R10+1
```

Aufgabe 9: Micro16 – CRC-Code Berechnung

Beim CRC-Verfahren muss zur Berechnung des Sicherungsanhangs, eine Polynomdivision mit Rest durchgeführt werden. Um den Sicherungsanhang zu einem binär gegeben Datenwort zu berechnen, wird dieses zunächst um n Stellen nach links verschoben (d.h. es muss mit x^n multipliziert werden), wobei n der Grad des Generatorpolynoms ist. Danach wird durch das Generatorpolynom dividiert und der dabei entstehende Rest beschreibt den Sicherungsanhang. Dies lässt sich mit einfachen Hardware-Bausteinen realisieren. Dazu verwendet wird üblicherweise

- ein Schieberegister mit n Bits und
- in Abhängigkeit des Generatorpolynoms eine Anzahl von XOR-Gattern ($\oplus$).

Für das Generatorpolynom $G(x) = x^5 + x^4 + x^2 + 1$ hat das Schieberegister mit seinen XOR-Gattern beispielsweise folgenden Aufbau:



Um den Sicherungsanhang für ein gegebenes Datenwort (z.B.: „11011“) zu berechnen, wird das nach links verschobene Datenwort (z.B.: „1101100000“) beginnend mit dem MSB durch das Schieberegister geschoben. Nachdem das LSB ins Schieberegister geschoben wurde, enthält das Schieberegister den Rest der Polynomdivision (z.B.: „00101“).

Die nachfolgende Tabelle zeigt den Inhalt des Schieberegisters wenn schrittweise das Datenwort „11011“ reingeschoben wird. In der letzten Zeile ist der Sicherungsanhang hervorgehoben.

5	4	3	2	1	Eingabebit des Datenworts
0	0	0	0	0	1
0	0	0	0	1	1
0	0	0	1	1	0
0	0	1	1	0	1
0	1	1	0	1	1
1	1	0	1	1	0
0	0	0	1	1	0
0	0	1	1	0	0
0	1	1	0	0	0
1	1	0	0	0	0
0	**0**	**1**	**0**	**1**	

Schreiben Sie ein Micro16-Programm, das die Berechnung des oben gegebenen Schieberegisters nachbaut. Das 5-Bit große Datenwort liegt dabei in Register R0 beginnend mit dem LSB vor. Am Ende der Berechnung soll der Sicherungsanhang in Register R0 beginnend mit dem LSB abgelegt werden (d.h. das LSB kommt als erstes ins Schieberegister).

Beispiel: R0: 0000 0000 0001 1011 $\Rightarrow$ R0 $\leftarrow$ 0000 0000 0000 0101
R0: 0000 0000 0000 1010 $\Rightarrow$ R0 $\leftarrow$ 0000 0000 0000 1001

```
# Zähler Variable
R9 ← lsh (1)
R9 ← lsh (R9)
R9 ← lsh (R9+1)

:loop

#LSB berechnen
R6 ← rsh (R0)
R6 ← lsh (R6)
R6 ← ~R6
R6 ← R6+1
R6 ← R0+R6

#R7 = R5 in 2's
R7 ← ~R5
R7 ← R7+1

#(R5')R10 = R5 ⊕ R4
(R4+R7); if z goto .SetR10to0
R8 ← ~R4
R8 ← R8+1
(R5+R8); if z goto .SetR10to0
R10 ← 1
:continueR10

#R4 = R3
R4 ← R3

#R3 = R5 ⊕ R2
(R2+R7); if z goto .SetR3to0
R8 ← ~R2
R8 ← R8+1
(R5+R8); if z goto .SetR3to0
R3 ← 1
:continueR3

#R2 = R1
R2 ← R1

#R1 = R5 ⊕ E
(R6+R7); if z goto .SetR1to0
R8 ← ~R6
R8 ← R8+1
(R5+R8); if z goto .SetR1to0
R1 ← 1
:continueR1

R5 ← R10

#LSB abschneiden
R0 ← rsh (R0)

#Zähler um eins runter
R9 ← R9+(-1)

#Wiederholen wenn Zähler ≠ 0
R9; if z goto .end
goto .loop

:SetR1to0
R1 ← 0
goto .continueR1

:SetR3to0
R3 ← 0
goto .continueR3

:SetR10to0
R10 ← 0
goto .continueR10

:end

R0 ← R5
R0 ← lsh (R0)
R0 ← R0+R4
R0 ← lsh (R0)
R0 ← R0+R3
R0 ← lsh (R0)
R0 ← R0+R2
R0 ← lsh (R0)
R0 ← R0+R1
```