2019W

4. *(_____ / 8 Punkte)* Micro16

Gegeben sind die folgenden Micro16-Instruktionen in binärer Notation:

	AMUX	COND	ALU	SH	MBR	MAR	RDWR	MS	ENS	S-BUS	B-BUS	A-BUS	ADR
A:	0	00	01	01	0	0	0	0	1	1010	0001	0001	0000 0000
B:	0	00	01	00	0	0	0	0	1	1010	0010	1010	0000 0000
C:	1	00	10	00	1	1	0	1	0	0000	1010	0000	0000 0000
D:	0	00	00	00	0	0	0	1	0	0000	0000	0000	0000 0000

Codierung	Beschreibung
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
R0	0100
R1	0101
...	...
R10	1110

(a) Geben Sie die Instruktion B in hexadezimaler Form an.

B: $(081A2A00)_{16}$

Um die Instruktion in eine Hexadezimale Form zu bringen, muss man die Bits der Instruktion von rechts nachts links lesen. Dies muss man immer in 4er Bloecken machen.

(b) Geben Sie die Instruktionen A, B, C und D in symbolischer Notation (Micro16-Code) an.
Hinweis: Es handelt sich um gültige Instruktionen.

```
A: R6 <- lsh(1 + 1)                          # R6 = 4
B: R6 <- R6 + (-1)                           # R6 = 3
C: MBR <- MBR & R6; MAR <- R6; wr;  # MBR = MBR & 3
D: wr;                                       # mem[3] = MBR & 3
```

Der Inhalt des MBR macht mit 3 ein Bitweises &, deswegen bleiben am Ende nur 2 Stellen übrig. $(3)_{10} = (11)_2$

Wenn 2 Bits übrig bleiben, dann kann das Ergebnis maximal 3 sein, daher ist es modulo 4 Operation.

(c) Welche mathematische Funktion realisiert dieses Programm?

Maskierung der letzten beiden Bits.
MBR = MBR mod 4

6. *(_____ / 10 Punkte)* Micro16-Programm

Vervollständigen Sie nachfolgendes Micro16-Programm, das den Befehl BFFFO (Bit Field Find First One) realisiert. Durch diesen Befehl kann die niederwertigste 1 in einem Bitfeld gesucht werden. Die Bitfolge befindet sich auf Speicheradresse 0x0000. Das Ergebnis der Suche soll auf Speicheradresse 0xFFFF als Zahl n ($0 \leq n \leq 16$) abgelegt werden. Wurde keine 1 gefunden, soll das Ergebnis auf Speicheradresse 0xFFFF mit dem Zahlenwert 0 belegt werden.

Beispiele:
Speicheradr. 0x0000: 0111 0100 1110 0000 ⟶ Speicheradr. 0xFFFF: 0000 0000 0000 0110
Speicheradr. 0x0000: 0000 0000 0000 0000 ⟶ Speicheradr. 0xFFFF: 0000 0000 0000 0000

```
00: MAR ← ____0____; rd
01: rd;
02: R0 ← ____MBR____;
03: R1 ← 0;
04: (R0) if ____Z____ goto ____10____;
05: R1 ← R1 + ____1____;
06: (R0 & ____1____) if Z goto ____08____;
07: goto ____10____;
08: R0 ←RSH(16);
09: goto ____05____;
10: MAR ← ____-1____; MBR ← ____R1____; wr;
11: wr;
```

2. *(_____ / 6 Punkte)* Micro16

Gegeben sind nachfolgende Bitfolgen A, B und C, 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	00	00	0	1	1	1	0	0000	0010	0000	0000 0000
B:	0	00	11	00	0	0	1	1	1	0100	0000	0001	0000 0000
C:	1	00	10	00	0	0	0	0	1	0100	0100	0000	0000 0000

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

Codierung	Beschreibung
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
R0	0100
R1	0101
...	...
R10	1110

Instruktion A: MAR <- (-1); rd

Instruktion B: rd; R0 <- ~(+1)

Instruktion C: R0 <- MBR & R0

8. *(_____ / 9 Punkte)* Micro16

Bei der arithmetischen Rechtsschiebeoperation (*arithmetic shift right*) werden Bits, wie folgt, nach rechts verschoben:

- Bits, die rechts hinausgeschoben werden, gehen verloren.
- Links werden Kopien des höchstwertigen Bits (MSBs) eingefügt.

Beispiele, bei denen arithmetisch um eine Stelle nach rechts verschoben wird:

R0 = 11001100 10010101 → R0 = 11100110 01001010
R0 = 01001100 10010101 → R0 = 00100110 01001010

Anmerkung: Üblicherweise wird angenommen, dass bei der arithmetischen Rechtsschiebeoperation das MSB das Vorzeichenbit ist und die Binärzahl im Zweierkomplement dargestellt ist. Wird dann arithmetisch um ein Bit nach rechts verschoben, so entspricht das der Division durch die Zahl 2, bei der die Rundungsfunktion *round towards negative infinity* verwendet wird.

Schreiben Sie ein Micro16-Programm, das den Inhalt von **R0** arithmetisch um genau ein Bit nach rechts verschiebt.

Micro-Code:

5. *(____ / 10 Punkte)* Schreiben Sie ein Micro16-Programm, das zwei Datenworte dividiert. Der Dividend liegt im Speicher an der Adresse 0xFFFE und der Divisor liegt im Register R0. Subtrahieren Sie dafür den Divisor so oft vom Dividenden, bis dieser negativ wird und erhöhen Sie dabei jedes mal das Ergebnisregister um 1. Am Ende des Programms soll das Ergebnis der Division im Register R1 stehen und der verbleibende Rest soll im Register R7 stehen. Sie können davon ausgehen, dass der Dividend und der Divisor beide positiv sind.

Micro-Code:

Dividend : Divisor

```
R3 ← (-1) + (-1)
MAR ← R3; rd
rd

R3 ← MBR
R1 ← 0
R0 ← ~R0
R0 ← R0+1

R10 ← 0
:while 1
R9 ← R9+1
R3 ← R0+R9; if N goto .endwhile
  R9
```

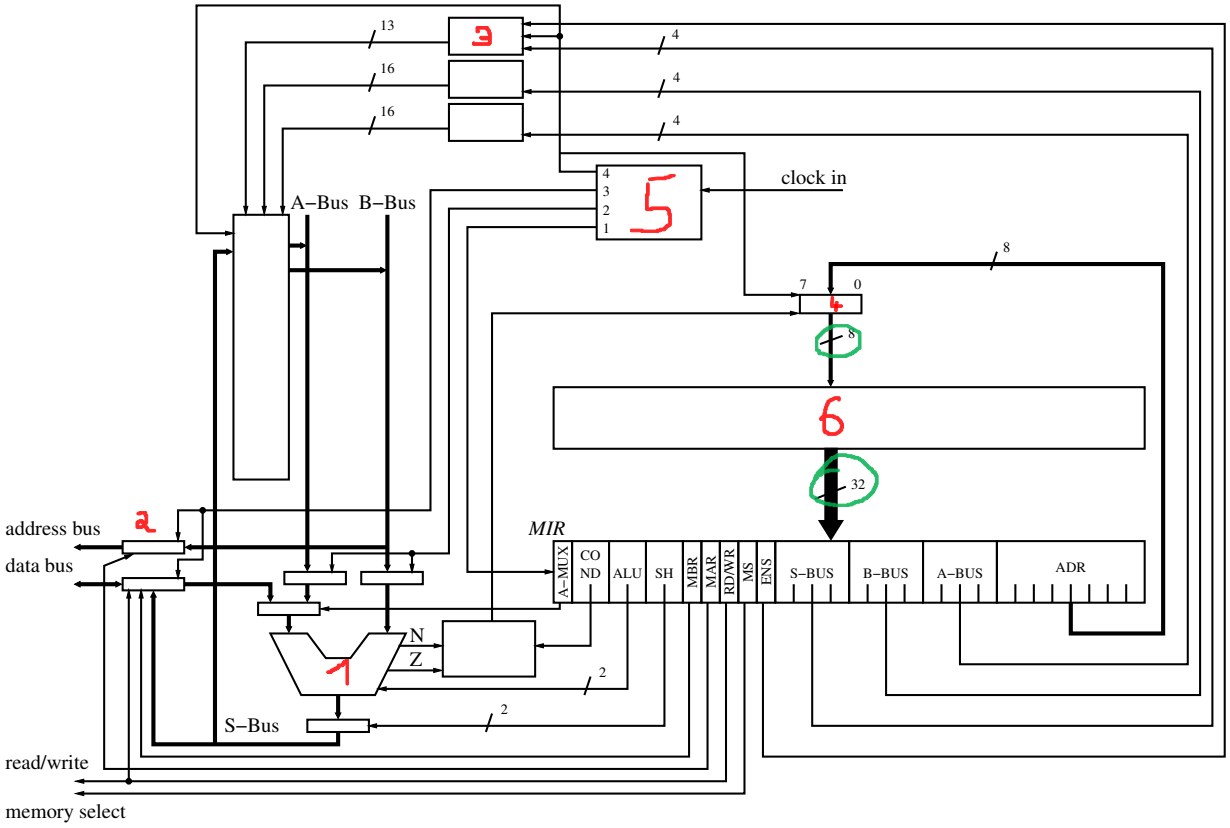
4. *(_____ / 8 Punkte)* Das nachfolgende Micro16-Programm soll überprüfen, ob das Datenwort, das im Register R0 abgelegt ist, gerade oder ungerade Parität hat. Ist die Anzahl der Einsen im Register R0 gerade, dann wird der Wert 0 im Register R1 abgelegt. Ansonsten wird der Wert +1 im Register R1 abgelegt.

   Vervollständigen Sie das Programm in den Zeilen 00, 03, 08 und 11.

   Micro-Code:

```
00:      R1 ← ____0____
01:   :loop
02:      (R0 ) if Z goto .check
03:      (R0 & 1) if ___z___ goto .noadd
04:      R1 ← R1 +1
05:      R0 ← rsh(R0 )
06:      goto .loop
07:   :noadd
08:      R0 ← __rsh(R0)__
09:      goto .loop
10:   :check
11:      (R1 & 1) if Z goto .even
12:   :odd
13:      R1 ← 1
14:      goto .end
15:   :even
16:      R1 ← 0
17:   :end
```

3. *(____ / 8 Punkte)* Nachfolgende Abbildung zeigt Ihnen den Aufbau des Micro16.



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

- Arithmetic Logic Unit (ALU): (1) ✓
- Memory Address Register (MAR): (2)
- S-Bus Decoder: (3) ✓
- Micro Instruction Counter (MIC): (4) ✓
- 4-Phase Clock (Control Unit): (5) ✓
- Micro-Programm-Speicher: (6) ✓

(b) **Erklären** Sie anhand der Abbildung die Größe des Micro-Programmspeichers.

$1^8 \times 32 =$

2^8 Adressen

32 Ausgangsbits

$2^8 \cdot 32$

3. *(____ / 10 Punkte)* Schreiben Sie ein Micro16-Programm, das den Inhalt von zwei 8 Byte großen Bereichen im Speicher vergleicht. Es kann davon ausgegangen werden, dass die Startadresse des ersten Speicherbereichs in Register R0 und die Startadresse des zweiten Speicherbereichs in Register R1 liegt. Die zu vergleichenden 8 Byte liegen von diesen Startadressen ausgehend, aufsteigend im Speicher. Falls der Inhalt der beiden Speicherbereiche bitweise ident ist, so soll der Wert +1 im Register R7 abgelegt werden. Ansonsten soll der Wert 0 in R7 abgelegt werden.

   Micro-Code:

Micro-Code:

R0: 0000 1111

R1: 0000 1111

```
PC ← lsh(1+1)
R7 ← 1
:loop
(PC); if Z goto .end
MAR ← R0; rd
rd
R5 ← MBR
R0 ← R0 + (-1)
MAR ← R1; rd
rd
R6 ← MBR
R1 ← R1 + (-1)
R6 ← ~R6
R6 ← R6 + 1
PC ← PC + (-1)
R5 + R6; if Z goto .loop
R7 ← 0
:end
```

2017W

-1 im Zweierkomplement ist 1111 1111 1111 1111. Wenn ich es right-shifte, dann ist es 0111 1111 1111 1111, was die groesste positive zahl ist. Umgewandelt in Hexadezimal ergibt das 0x7FFF

4. *(_____ / 10 Punkte)* Sie haben folgendes Micro16-Programm gegeben:

```
01: AC <- rsh(-1);
02: AC <- AC + 1;
03: MAR <- AC; rd;
04: rd;
05: :loop
06:   (MBR & 1) if Z goto .zero
07:   R10 <- R10 + 1;
08:   :zero
09:   (MBR <- rsh(MBR)) if Z goto .end
10: goto .loop
11: :end
12: R10 <- R10 & 1;
13: MBR <- R10;
14: AC <- AC + 1;
15: MAR <- AC; wr;
16: wr;
```

AC <- 0x7FFF

AC <- 0x8000

read from address 0x8000

if the result is zero, skip the next instruction

right-shift the mbr.

AC <- 0x8001

Write to the new address

mit einem & wird eine logische Verundung gemacht. Das Ergebnis ist so gross, wie der rechte Operand. Bei MBR & 1 wird geschaut, ob das letzte Bit des MBR 1 ist.

Shifting takes place after the check if the result is zero or negative. So the result of the shift is not taken into account when performing the jump.

(a) Das gegebene Micro16-Programm liest zu Beginn einen Wert vom externen Speicher, führt eine Berechnung durch und schreibt das Ergebnis wieder in den Speicher. Von welcher Speicheradresse wird gelesen und auf welche Adresse wird geschrieben? Geben Sie die Adressen in hexadezimaler Notation an.

Lesen: 0x8000

Schreiben: 0x8001

(b) Tragen Sie die Registerinhalte von `MBR` und `R10` binär direkt vor jeder Ausführung von `loop:` in die nachfolgende Tabelle ein. Sobald das Programm terminiert, lassen Sie nachfolgende Tabellenzeilen frei. Führende Nullen können weggelassen werden.

`loop`	Register `MBR`	Register `R10`
0	0000 0000 0000 1100	0000 0000 0000 0000
1	0000 0000 0000 0110	0000 0000 0000 0000
2	0000 0000 0000 0011	0000 0000 0000 0000
3	0000 0000 0000 0001	0000 0000 0000 0001
4	0000 0000 0000 0000	0000 0000 0000 0011
5		

(c) Das gegebene Micro16-Programm überprüft, ob in einem empfangenen, im Speicher befindlichen Codewort ein erkennbarer Fehler aufgetreten ist. Falls in R10 am Ende des Programms 1 steht, so ist kein erkennbarer Fehler aufgetreten. Welche Methode zur Codewortabsicherung wurde beim Sender verwendet?

Odd Parity

(d) Um zuverlässig auch nach wiederholter Ausführung das richtige Ergebnis zu liefern, fehlt dem gegebenen Micro16-Programm eine Instruktion. Welche Instruktion fehlt und zwischen welchen beiden Zeilen würden Sie diese Instruktion einfügen?

R10 wird nicht intialisiert, daher wissen wir nicht, was in R10 steht und sollte deshalb vor der loop mit 0 initialisiert werden.

9. *(10 Punkte)* Übersetzen Sie die nachfolgenden zwei Micro16-Instruktionen C und D in Micro16-Code! Bei den Instruktionen handelt es sich um korrekte Micro16-Instruktionen.

Wenn MS (Memory Select) auf 0 gesetzt wurde, dann ist es egal, was in RDWR steht. 0 in RDWR steht fuer write, aber da MS 0 ist, ist es egal.

Instruktion A: `R1 ← ~R1`

Instruktion B: `MBR ← ~MBR`

Instruktion C: `MBR ← R1 & MBR`

Instruktion D: `MAR ← (-1); wr`

Instruktion E: `wr`

Codierung	Beschreibung
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
R0	0100
R1	0101
...	...
R10	1110

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

	AMUX	COND	ALU	SH	MBR	MAR	RDWR	MS	ENS	S-BUS	B-BUS	A-BUS	ADR
C:	1	00	10	00	1	0	0	0	0	0000	0101	0000	0000 0000
D:	0	00	00	00	0	1	0	1	0	0000	0010	0000	0000 0000

(b) Auf welche externe Speicheradresse wird in der Micro16-Instruktion D zugegriffen?

1111 1111 1111 1111 0xFFFF

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

$\neg R1 \wedge \neg MBR = \neg(R1 \vee MBR) = NOR$

9. *(12 Punkte)* Schreiben Sie ein Micro16-Programm, das prüft, ob zwei Bitfolgen, die in den Registern `R1` und `R2` abgelegt sind, ident sind. Sind sie ident, soll am Ende im Register `R0` die Zahl 0 hinterlegt sein, anderenfalls die Zahl 1.

```
R2 <- ~R2
R2 <- R2 +1
R0 <- R1 + R2; if Z goto .end
R0 <- 1
:end
```

Idee:
Um zu ueberpruefen, ob zwei Zahlen ident sind, muss man die beiden Zahlen subtrahieren. Da der Micro16 nicht subtrahieren kann, muss man das Zweierkomplement einer der beiden Zahlen bilden. Dadurch ist die Zahl negativ und kann addiert werden. Wenn das Ergebnis der Subtraktion 0 ist, dann sind die Zahlen gleich und das Programm kann beendet werden.

Platz für Notizen: