1. In der folgenden Tabelle finden Sie häufig verwendete Zahlensysteme. Füllen Sie die Tabelle vollständig aus, indem Sie aus den gegebenen Zahlen jeweils die fehlenden Darstellungen berechnen.

Binärzahl	Oktalzahl	Dezimalzahl	Hexadezimalzahl
110			
	57		
		167	
			3C

Lösung:

Binärzahl	Oktalzahl	Dezimalzahl	Hexadezimalzahl
110	6	6	6
101111	**57**	47	2f
10100111	247	**167**	a7
111100	74	60	**3C**

2. **Fließkommazahlen**

Eine 16-Bit-Fließkommazahlendarstellung habe folgende Eigenschaften:

- Das Vorzeichen werde in einem Bit dargestellt.
- Die Mantisse habe 10 Bit.
- Der Exponent habe 5 Bit und ein Bias von 15.

(2.1) Stellen Sie die folgenden zwei Zahlen in Fließkommadarstellung dar:

- 110,5
- 0,0625

Lösung:

$$110,5 = (1101110,1)_2 = 1,1011101 \cdot 2^6$$

$$0,0625 = (0,0001)_2 = 1,0 \cdot 2^{-4}$$

Nutzen Sie diese Vorlage zur Darstellung der ersten Zahl:

0				4							14				15
VZ	Mantisse										Exponent				

Lösung:

0				4							14				15
VZ	Mantisse										Exponent				
0	1	0	1	1	1	0	1	0	0	0	1	0	1	0	1

Nutzen Sie diese Vorlage zur Darstellung der zweiten Zahl:

0				4							14				15
VZ	Mantisse										Exponent				

Lösung:

0				4							14				15
VZ	Mantisse										Exponent				
0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1

(2.2) Addieren Sie nun beide Zahlen nach der Additionsmethode für Fließkommazahlen und stellen Sie das Ergebnis Fließkommazahl dar:

Lösung:

kleineren Exponenten anpassen:

$$1,1011101 \cdot 2^6 + 1,0 \cdot 2^{-4} = 1,1011101 \cdot 2^6 + 0,0000000001 \cdot 2^6 = 1,1011101001 \cdot 2^6$$

Nutzen Sie diese Vorlage zur Darstellung des Ergebnisses:

0	4										14	15
VZ	Mantisse										Exponent	

Lösung:

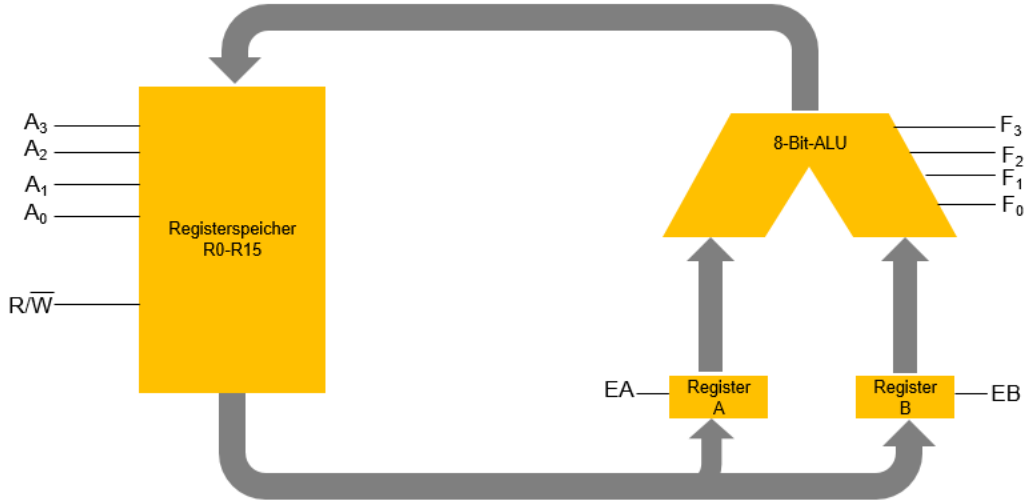
0	4										14	15
VZ	Mantisse										Exponent	
0	1	0	1	1	1	0	1	0	0	1	1	0	1	0	1

3. Implementieren Sie ein XOR-Gatter mit den drei Eingängen A, B und C, das die folgende Funktion realisiert:

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Verwenden Sie ausschließlich AND-, OR-, und NOT-Gatter. AND- und OR-Gatter dürfen drei Eingänge haben.

4. Gegeben sei das folgendes aus der Vorlesung bekanntes Rechenwerk mit Registerspeicher:



Signal	Beschreibung
EA	Enable A (Register A übernimmt Wert vom Eingang)
EB	Enable B (Register B übernimmt Wert vom Eingang)
A_3, A_2, A_1, A_0	Adresse am Registerspeicher (eines der Register 0 bis 15 wird adressiert)
$R/\overline{W}$	Lese aus dem Registerspeicher ($R/\overline{W} = 1$) bzw. schreibe in den Registerspeicher ($R/\overline{W} = 0$)
F_3, F_2, F_1, F_0	ALU-Kontrollsignale (s. nächste Tabelle)

ALU-Kontrollsignale:

F_3	F_2	F_1	F_0	Ergebnis
0	0	0	0	0
0	0	0	1	A
1	0	0	1	NOT A=$\overline{A}$
0	1	0	0	$A \wedge B$
0	1	0	1	$A \vee B$
0	1	1	0	$A \oplus B$
0	1	1	1	$A + B$
1	1	1	1	$B - A$

Implementieren Sie nun die einzelnen Schritte eines mikroprogrammierten Steuerwerks, um die folgende Funktion zu realisieren:

$R2 = \overline{R0 + R1}$

Nutzen Sie bitte folgende Tabelle:

Beschreibung	$A_3A_2A_1A_0$	$R/\overline{W}$	EA	EB	$F_3F_2F_1F_0$

Lösung:

Beschreibung	$A_3A_2A_1A_0$	$R/\overline{W}$	EA	EB	$F_3F_2F_1F_0$
R0 laden	0000	1	1	0	xxxx
R1 laden	0001	1	0	1	xxxx
Addieren und in R2 speichern	0010	0	0	0	0111
R2 in A laden	0010	1	1	0	xxxx
A negieren und in R2 speichern	0010	0	0	0	1001

5. Gegeben sei folgendes kleines Code-Fragment in der Programmiersprache C.

```
int fibonacci(int n) {
    if(n == 0){
        return 0;
    } else if(n == 1) {
        return 1;
    } else {
        return (fibonacci(n-1) + fibonacci(n-2));
    }
}
```

Gegeben sei außerdem eine Stack-Maschine mit dem folgenden Instruktionssatz:

Instruktion	Beschreibung
PUSH #const	schreibt die Konstante const auf den Stack (SP = SP + 1; [SP] = const)
POP	nimmt oberstes Stack-Element (SP = SP - 1)
SWAP	tausche obersten beiden Stackelemente
DUP	dupliziere oberstes Stackelement (SP = SP + 1; [SP] = [SP - 1])
DUP2	dupliziere zweites Stackelement (SP = SP + 1; [SP] = [SP - 2])
INC	erhöhe oberstes Stack-Element ([SP] = [SP] + 1)
DEC	verringere oberstes Stack-Element ([SP] = [SP] - 1)
ADD	addiere die obersten Stack-Elemente und schreibe Ergebnis auf Stack ([SP-1] = [SP-1] + [SP]; SP = SP - 1)
CMP, SUB	subtrahiere die obersten Stack-Elemente und schreibe Ergebnis auf Stack ([SP-1] = [SP-1] - [SP]; SP = SP - 1)
MOD	berechne Modulo-Operation mit obersten beiden Stack-Elementen und schreibe Ergebnis auf den Stack ([SP-1] = [SP-1] mod [SP]; SP = SP - 1)
JMP label	Springe zum Ziellabel label
JZ label	Springe zum Ziellabel label, wenn Element auf Stack null; SP = SP - 1
JNZ label	Springe zum Ziellabel label, wenn Element auf Stack nicht null; SP = SP - 1
JL label	Springe zum Ziellabel label, wenn Element auf Stack kleiner null; SP = SP - 1
CALL label	Rufe Unterfunktion auf; SP = SP + 1; [SP] = PC (Rücksprungadresse); PC = label
RET	Rückkehr zum aufrufenden Programm; [PC] = [SP]; SP = SP - 1

Implementieren Sie das C-Programm in Assembler auf der Stack-Maschine.

Lösung:

```
fibonacci:
        SWAP
        DUP
        JZ n_null
        DUP
        DEC
        JZ n_eins
        DUP
        DEC
        CALL fibonacci
        DUP
        DEC
        CALL fibonacci
        ADD
        JMP ende
n_eins: PUSH #1
        JMP ende
n_null: PUSH #0
ende:   SWAP
        RET
```

6. Gegeben sei das folgende Programm in einer Assemblersprache.

```
        MOV     R3, vars
        LOAD    R0, [R3]
        LOAD    R1, [R3+1]
        ADD     R0, R1    # R0 = R0 + R1
        PUSH    R0
        CALL    println
        JMP     end

vars    db      10, 15
end:
```

Gegen sei ein Prozessor mit der aus der Vorlesung bekannten 5-stufigen Pipeline IF, ID, OF, EX, WB.

(6.1) Identifizieren Sie für jeden Ihnen bekannten Pipelinekonflikt ein Beispiel in dem Code. Markieren Sie dieses Beispiel und beschriften Sie den Pipelinekonflikt.

(6.2) Welche Maßnahmen können gegen die unterschiedlichen Typen von Pipelinekonflikten unternommen werden?

Lösung:

- Ressourcenkonflikt: z. B. zwischen LOAD (Z.2) und ADD, da LOAD im OF auf den Speicher zugreift, um Wert von Adresse R3 zu laden und ADD im IF geladen wird; Gegenmittel: z. B. Harvard-Architektur
- Datenabhängigkeit: z. B. Zeile 1 und 2 (R3 wird in Z. 2 im OF benötigt, in Z. 1 aber erst im WB geschrieben); Gegenmaßnahmen: NOPs einfügen, Befehle umsortieren, Bypassing
- Kontrollflusskonflikt: z. B. kennt CALL das Sprungziel erst im EX - daher werden falsche Folgeinstruktionen in die Pipeline geladen; Gegenmaßnahmen: Branch Prediction, Sprungvorverlegung, Delayed Branch

7. Gegeben sei ein Computer mit einer speicherbasierten Anbindung von I/O-Geräten. Der Rechner habe einen 16-Bit Adressbus und die Adressbereiche der vorhandenen I/O-Geräte sind im Adressdekoder wie in der folgenden Tabelle definiert:

I/O-Gerät	Startadresse	Adress-Maske
Hauptspeicher	0x0000	0x8000
Grafikkarte	0x8000	0xC000
Tastatur	0xC000	0xFF00
Maus	0xC100	0xFF00
Soundkarte	0xF100	0xFF00

Die Startadresse definiere die Startadresse im 16-Bit-Adressbereich des Computers und die Adress-Maske spezifiziert mittels Nullen in der Bitmaske, welcher Adressanteil die Register im Gerät adressiert (Einsen spezifizieren entsprechend den Startadressteil).

Ein Programm greife nun auf die folgenden Speicheradressen zu:

Adresse	Gerät
0xF140	
0xC240	
0x0000	
0x7FFE	
0xA010	

Ergänzen Sie in der Tabelle, zu welchem I/O-Gerät die jeweilige Speicheradresse gehört. Begründen Sie kurz.

Lösung:

Adresse	Gerät
0xF140	0xF140 AND 0xFF00 = 0xF100, also Soundkarte
0xC240	0xC240 AND 0xFF00 = 0xC200 0xC240 AND 0xC000 = 0xC000 0xC240 AND 0x8000 = 0x8000, passt also kein Gerät
0x0000	0x0000 AND 0x8000 = 0x0000, also Hauptspeicher
0x7FFE	0x7FFE AND 0x8000 = 0x0000, also Hauptspeicher
0xA010	0xA010 AND 0xC000 = 0x8000, also Grafikkarte

8. Wo liegen die Unterschiede zwischen einem Unterprogrammaufruf (JSR) und einem Interrupt, der eine Interruptservice-Routine aufruft? Welche Register müssen jeweils gesichert werden?