C-ÜBUNGSAUFGABEN FUNKTIONEN-OUTPUT 1

Übungen zur Digitaltechnik

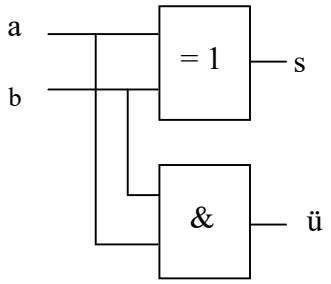
1) Halbaddierer

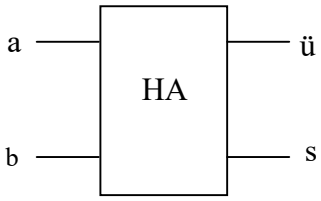
Bemerkung:

In bestimmten Programmiersprachen (z.B. C, Java) gibt es für ganzzahlige Datentypen u.a. die folgenden bitweise Operatoren, mit denen man z,B, Schaltungen aus der Digitaltechnik simulieren kann:

& (logische UND), | (logische ODER bzw. >=1), ^ (logische exklusive ODER bzw. = 1)

Links unten befindet sich die Digitalschaltung eines Halbaddierers und rechts davon sein Blockschaltbild.





a) Erstellen Sie die Funktion

```
void halbaddierer(int a, int b, int *s, int *uebertrag)
```

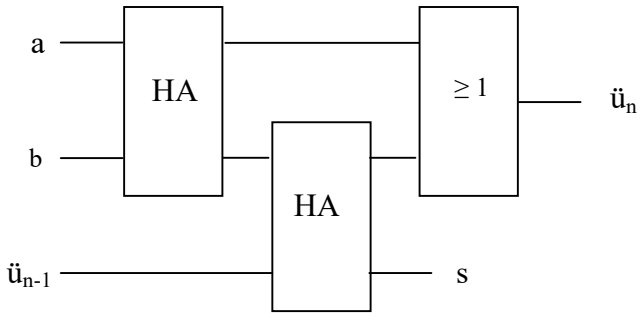
b) Erstellen Sie die Funktion

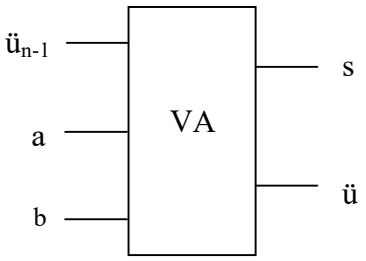
```
void printHalbaddierer()
```

die die Wertetabelle eines Halbaddierers auf dem Bildschirm ausgibt.

2) Volladdierer

Links unten befindet sich die Digitalschaltung eines Volladdierers und rechts davon sein Blockschaltbild.





a) Erstellen Sie die Funktion

```
void volladdierer(int a, int b, int uebertragIn, int *s, int *uebertragOut)
```

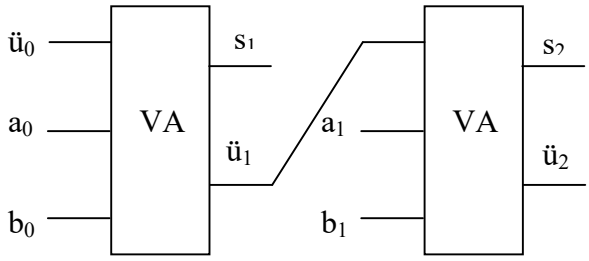
b) Erstellen Sie die Funktion

```
void printVolladdierer()
```

die die Wertetabelle eines Volladdierers auf dem Bildschirm ausgibt.

3) 2-Bit-Volladdierer

Links unten befindet sich die Digitalschaltung eines 2-Bit-Volladdierers



Erstellen Sie die Funktion
void zweiBitVolladdierer(...)

4) n-Bit-Volladdierer

Wenn man den 2-Bit-Volladdierer verallgemeinert, kommt man zum n-Bit-Volladdierer.
a) Erstellen Sie die Funktion
void nBitVolladdierer (int a[], int b[], int ergebnis[])
wobei in a[0], b[0], ergebnis[0] jeweils die Anzahl der Bits stehen muss, also:
a[0] = b[0] = n und ergebnis[0] = n+1

b) Erstellen Sie die Funktion
void print_nBitVolladdierer (int a[], int b[], int ergebnis[])
die die Wertetabelle eines n-Bit-Volladdierer auf dem Bildschirm ausgibt.

c) Erstellen Sie die Funktion
int test_4Bit_Paralleladdierer()
die die Korrektheit eines 4-Bit-Volladdierer testet:
Für alle möglichen Werte für a (0000-1111) und für b (0000-1111) soll nachprüft werden, ob der 4-Bit-Volladdierer korrekt arbeitet, in dem die Eingänge bzw der Ausgang in eine Dezimalzahl verwandelt.

Beispiel:
0011 + 0001 gibt mit dem 4-Bit-Volladdierer den Wert 0100
Umwamdlung:
0011 --> 3
0001 --> 1
0100 --> 4
3 + 1 ergibt dezimal 4. Also ist der 4-Bit-Volladdierer für diese 2 Eingangswerte korrekt.

5) Simulation eines selbst erfundenen, fiktiven Mikroprozessors FMP

Beschreibung

B1) Arbeitsspeicher und Register

Der FMP ist mit dem FAS (fiktiven Arbeitsspeicher) durch eine Daten- und Adressleitung verbunden. Der FAS besteht aus 256 Bytes. Jedes einzelne Byte besitzt eine Adresse und kann über diese Adresse vom FMP angesprochen (adressiert) werden.

Adr	Inhalt
0	
1	
2	
4	
...	
...	
...	
255	

Zusätzlich zu diesem FAS gibt es noch 3 schnelle Speicher, die so genannten Register AX, BX, IP (jeweils 1 Byte groß), die sich direkt im FMP befinden.
Der IP (Instruction Pointer, deutsch Befehlszähler) ist ganz wichtig für die Steuerung der Befehlsabarbeitung. In ihm steht immer die Adresse des nächsten auszuführenden Maschinenbefehls.

AX	
BX	
IP	

B2) Maschinenbefehl und Assemblerbefehl

Jedes Programm besteht auf unterster Ebene aus Maschinenbefehlen.
Maschinenbefehle bestehen aus 0 en und 1 en und stehen ab der Adresse 0 hintereinander im Arbeitssspeicher. Für den Menschen ist ein Maschinenbefehl schwer lesbar, deshalb gibt es für ihn eine symbolische Darstellung, den Assemblerbefehl.
Das Register IP hat zu Programmbeginn immer den Wert 0 und zeigt damit auf den nächsten auszuführenden Maschinenbefehl (an der Adresse 0).

Bemerkungen (zu den Assemblerbefehlen unten):
a steht für eine 1 Byte große positive Ganzzahl (0-255) und ist eine Adresse im FAS.
w steht für eine 1 Byte große positive Ganzzahl (0-255) und ist ein Wert, keine Adresse
d steht für eine 1 Byte große vorzeichenbehaftete Ganzzahl (-128 bis 127) und ist ein Wert (Distanz) und keine Adresse

MOV kopiert den rechten Operanden in den linken Operanden.
ADD addiert zum linken Operanden den rechten Operanden und speichert diese Summe im linken Operanden.
JBX w, d macht folgendes: wenn der Inhalt des Registers BX kleiner als dem Wert w ist, wird IP um den Wert (Distanz) d erhöht (d wird als einen Ganzzahl (-128 bis 127) interpretiert), wobei bei jedem Maschinenbefehl zuerst der Befehlszähler um die Befehlslänge erhöht wird.

B3) Die Menge der Maschinenbefehle des FMP (FMP-Befehlssatz):

Assemblerbefehl	Maschinenbefehl			Kurzbeschreibung
MOV AX, [a]	1	a		kopiere Inhalt von a nach AX
MOV [a], AX	2	a		kopiere AX an die Adresse a
MOV AX, w	3	w		kopiere w nach AX
MOV BX, w	4	w		kopiere w nach BX
ADD AX, w	5	w		AX + w --> AX
ADD AX, [a]	6	A		AX + [a] --> AX
ADD BX, w	7	W		BX + w --> BX
ADD BX, [a]	8	a		BX + [a] --> BX
JBX w, d	9	w	d	BX<w ==> IP + 2 + d --> IP
HLT	ED			FMP beendet Befehsabarbeitung

Man erkennt, daß alle Maschinenbefehle aus 2 Bytes bestehen außer HLT (1 Byte) und JBX w, d (3 Byte).

B4) Schema der Abarbeitung der Maschinenbefehle
Die Reihenfolge der Ausführung der einzelnen Befehle geschieht also nach folgendem sich wiederholenden Schema:
Schritt1:
Maschinenbefehl ausführen, desse Anfangsadresse im Befehlszähler IP steht, wobei der Maschinenbefehl zuerst den IP um die Befehlslänge erhöht.
Zusätzlich wird bei JBX w, d dann noch zu IP der Wert d dazuaddiert (wenn d negativ ist, kann man dadurch eine Schleife realisieren !).
Falls der Maschinenbefehl nicht im Befehlssatz des FMP vorkommt, stellt dieser die Befehlsabarbeitung ein und bringt eine entsprechende Fehlermeldung auf den Bildschirm.

Schritt2:
weiter mit Schritt1

B5) Beispiel:
alle Werte und Adressen in hexadezimaler Schreibweise.

0	03	MOV AX, 0
1	00	
2	04	MOV BX, F
3	0F	
4	05	ADD AX, 5
5	05	
6	07	ADD BX, 1
7	01	
8	09	JBX, 11, F9
9	11	
A	FA	
B	02	MOV [A0], AX
C	A0	
D	ED	HLT
E		
...		
A0		
...		

Programmablauf:
A1) IP = 0
Die CPU veranlaßt, daß der Inhalt der Adresse, auf die IP zeigt, also hier 03 in die CPU gebracht wird. Sie schaut in ihrer Befehlssammlung nach und interpretiert diese 03 als Anfang der Befehls MOV AX, 0. Dann führt sie diesen Maschinenbefehl aus (wobei dadurch IP um die Beefhlslänge 2 erhöht wird), also:
AX = 0 , IP = 2

A2) IP = 2
An der Adresse 2 steht der Assemblerbefehl MOV BX, F.
Dieser erhöht IP um 2 und kopiert in BX den Wert F, also:
BX = F, IP = 4

A3) IP = 4
An der Adresse 4 steht der Assemblerbefehl ADD AX, 5
Dieser erhöht IP um 2 und addiert zum Wert von AX noch 5 dazu, also:
AX = 5, IP = 6

A4) IP = 6
An der Adresse 6 steht der Assemblerbefehl ADD BX, 1
Dieser erhöht IP um 2 und addiert zum Wert von BX noch 1 dazu, also:
BX = F+1 = 10, IP = 8

A5) IP = 8
An der Adresse 8 steht der Assemblerbefehl JBX 11, F9
Dieser erhöht zunächst IP um die Befehlslänge 3 auf B.
Da BX = 10 < 11, wird dann außerdem noch zu IP, also B der Wert F9 dazuaddiert.
Da F9 dezimal -7 bedeutet, wird also zu B (dezimal 11) der Wert -7 addiert, was den Wert 4 ergibt. Also hat IP den Wert 4.
IP = 4

A6) IP = 4
An der Adresse 4 steht der Assemblerbefehl ADD AX, 5
Dieser erhöht IP um 2 und addiert zum Wert von AX noch 5 dazu, also:
AX = 5+5 = A, IP = 6

A7) IP = 6
An der Adresse 6 steht der Assemblerbefehl ADD BX, 1
Dieser erhöht IP um 2 und addiert zum Wert von BX noch 1 dazu, also:
BX = 10+1 = 11, IP = 8

A8) IP = 8
An der Adresse 8 steht der Assemblerbefehl JBX 11, F9
Dieser erhöht zunächst IP um die Befehlslänge 3 auf B.
Da BX = 11 < 11 nicht gilt, wird nichts mehr weiter gemacht, also:
IP = B

A9) IP = B
An der Adresse A steht der Assemblerbefehl MOV [A0], AX.
Dieser erhöht IP um 2 und kopiert AX an die Adresse A0, also:
[A0] = A und IP = D
A10) IP = D
An der Adresse D steht der Assemblerbefehl HLT.
Dieser erhöht IP um 2 und veranlaßt die CPU ihre Befehlsabarbeitung einzustellen.
IP = F

Aufgaben:
Nr1) Simulieren Sie diesen FMP durch ein Programm.

Nr2) Erstellen Sie einen von Ihnen gebastelten FMP und simulieren diesen.

Nr3) Schreiben Sie ein FMP-Programm, das alle seine Maschinenbefehle im FAS überschreibt.

Nr4) Schreiben Sie ein FMP-Programm, das einen "Maschinenbefehl" enthält, der nicht in der FMP-Befehlssammlung vorkommt.