RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

LEHRSTUHL INFORMATIK XI SOFTWARE FÜR EINGEBETTETE SYSTEME

Professor Dr.-Ing. Stefan Kowalewski
Hilal Diab, M.Sc.
Kamal Barakat, M.Sc
Dipl.-Inform. Dominik Franke

Aachen, 22. Januar 2010
SWS: V4/Ü2, ECTS: 7

Einführung in die Technische Informatik

WS 2009/2010

Blatt 12: VHDL,von-Neumann,PLA,MMIX

Ihre Lösung zu den mit (⋆) gekennzeichneten Übungen sollen Sie am **29.01.2010** in der Übung abgeben. Die Bearbeitung der Aufgaben in Lerngruppen von etwa drei oder vier Personen ist sinnvoll. Bitte geben Sie nur eine Lösung pro Lerngruppe ab.

Aufgabe 1: VHDL

a) Gegeben sei folgender Ausschnitt aus einer VHDL Spezifikation:

```
entity E2 is
  port(x: in std_logic_vector(3 downto 0);
    clk: in std_logic;
    f: out std_logic;
    g: out std_logic);
end E2;
```

Erstellen Sie ein Blockdiagramm (ähnlich Aufgabe 5), welches das Modul E2 und seine Ein- und Ausgänge darstellt.

b) Das Verhalten des Moduls E2 ist in folgendem VHDL Code spezifiziert:

```
architecture P3 of E2 is
begin
  process(x, clk)
  begin
    if rising_edge(clk) then
      if(x = "1100" or x = "0110") then
        f <= '1';
      else
        f <= '0';
      end if;
    end if;
  end process;
  g <= (x(3) and x(2)) or (x(1) and x(0));
end P3;
```

Welche Funktionen werden an den Ausgängen f und g des Moduls realisiert? Was haben sie gemeinsam und worin unterscheiden sie sich?

Aufgabe 2: (⋆)von-Neumann-Rechnermodell
1946 wurde das von-Neumann-Rechnermodell vorgestellt, das die Rechnerarchitektur bis heute maßgeblich beeinflusst. Arbeiten Sie die grundlegenden Organisationsprinzipien und Besonderheiten dieses Modells heraus, indem Sie folgende Fragen möglichst prägnant und in eigenen Worten beantworten. Zur Hilfe können Sie das Lehrbuch von Oberschelp/Vossen oder das Internet nehmen.

a) Mit dem von-Neumann-Rechnermodell wurde erstmalig das Konzept für einen echten 'general-purpose Computer' vorgeschlagen. Was ist darunter zu verstehen?

b) Das von-Neumann-Rechnermodell setzt sich aus drei Hauptbestandteilen zusammen. Welche Bestandteile sind dies und welchem Zweck dienen sie?

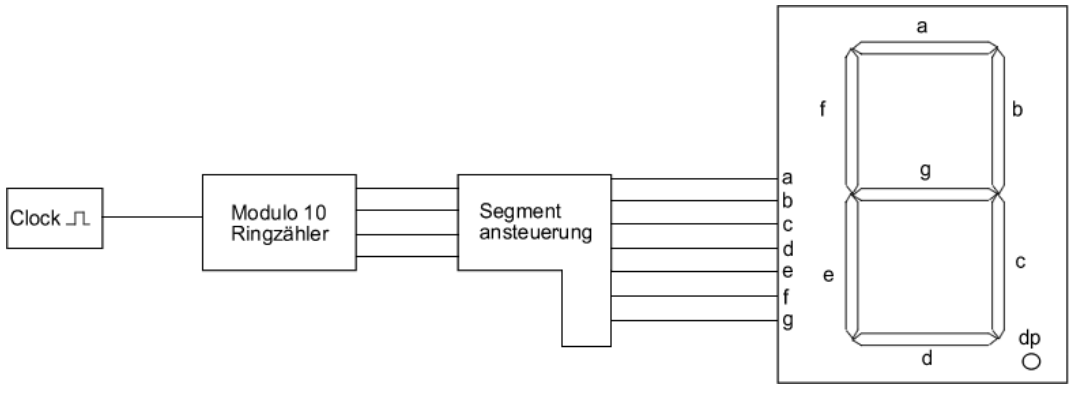
c) Im von-Neumann-Rechnermodell ist der Datenprozessor ein Bestandteil der CPU. Welche Aufgaben werden von welchen Komponenten dieses Prozessors erfüllt?

d) Das von-Neumann-Rechnermodell unterscheidet zwischen Daten- und Adressbus.Warum macht das Sinn? Es ergeben sich auch Zusammenhänge zwischen der Größe (in Bits) des MAR, des MBR, des Speichers, einer Speicherzelle sowie der Speicherzellenanzahl. Welche sind dies?

e) Bahnbrechend neu am von-Neumann-Rechnermodell war das Konzept einer quasi universellen Programmierbarkeit. Erörtern Sie in diesem Zusammenhang die Begriffe Maschinencode, Assemblersprachen sowie Ein- und Mehr-Adress-Befehle.

f) Charakteristisch für das von-Neumann-Rechnermodell ist ein Zwei-Phasen-Konzept der Befehlsverarbeitung. Welches Problem wird damit auf welche Weise gelöst?

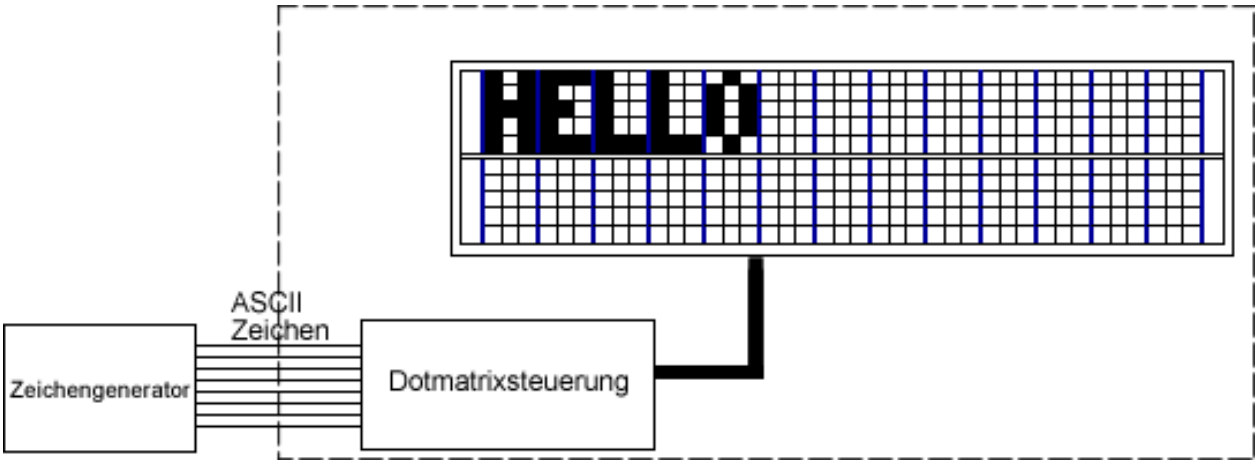
Aufgabe 3: 7 Segmentanzeige
Gegeben sei folgende Schaltung zur Ansteuerung eines 7-Segmentdisplays. Der Modulo-10-Ringzähler gibt Zahlen von 0-9 aus, wobei der Ringzähler nach jedem Takt inkrementiert wird. Die aus dem Ringzähler gewonnene Zahl in BCD-Darstellung, wird an eine Segmentansteuerungsschaltung weitergeleitet. Dort wird die Zahl so umgewandelt, dass die jeweilige Repräsenation der BCD-Zahl auf der Segmentanzeige sichtbar wird. Die Segmentanzeige besteht aus 7 Segmenten (inklusive Punkt, der hier nicht betrachtet wird). Jedes Segment kann unabhängig vom anderen durch Anlegen einer logischen 1 bzw. 0 am entsprechenden Eingang an- und ausgeschaltet werden.

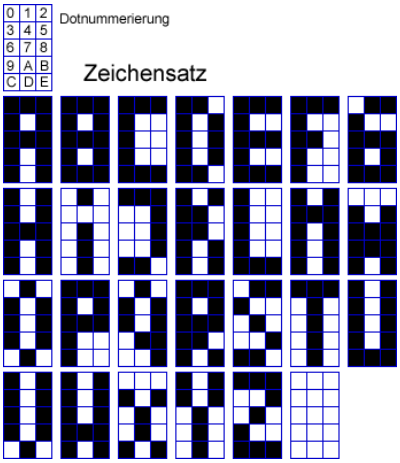


a) Stellen Sie eine Boolesche Funktion für den Ringzähler auf und minimieren Sie diese mit Hilfe von Karnaugh-Diagrammen. Zeichnen Sie ein Schaltnetz für diese Funktion (Leiterbahnen beschriften!). Alle Bauteile sollen einen Fan-In von 2 besitzen. Welche Bauteile werden benötigt, um den Ringzähler zu realisieren?
(Tipp: Es gibt Eingabewerte in der Funktion, für die keine Ausgabewerte definiert sind.)

b) Schreiben Sie die Funktionstabelle für die Segmentansteuerungsanzeige auf und minimieren Sie jede Funktion mit Hilfe von Karnaugh-Diagrammen.

Aufgabe 4: (⋆)Punktmarixdisplay
Ein Punktmatrixdisplay ist eine Anzeige zum Darstellen von Zeichen. Die Zeichen werden mit Hilfe einer Punktmatrix dargestellt, in der, je nach Buchstabe, ein Punkt an bzw. ausgeschaltet wird. Das Display hat Eingänge zum Empfangen der Zeichen bzw. zur Steuerung des Cursors (Cursorsteuerug wird hier nicht betrachtet). Das Punktmatrixdisplay verfügt über eine Punktmatrixsteuerung, die ankommende ASCII-Zeichen in ihre entsprechende Matrixvariante umwandelt - der Zeichensatz (alle Buchstaben sind Großbuchstaben!) des Displays ist unten angegeben. Ein 3x5 Punktmatrixdisplay mit einem angeschlossenen Zeichengenerator:





(Bsp.: Wenn das Zeichen A ausgegeben werden soll, müssen die Punkte 0,1,2,3,5,6,7,8,9,B,C,E auf logisch 1 gesetzt werden, während die Punkte 4, A, D auf logisch 0 gesetzt werden müssen.)

a) Realisieren Sie die Schaltfunktion der Punktmatrixsteuerung, die eine ASCII-Bitfolge in eine Bitfolge zum An- und Ausschalten der Punkte abbildet, mittels eines PLAs. Überlegen Sie dabei, wie viele Eingänge, Ausgänge und Spalten das PLA haben soll. (Tipp: Minimieren von Funktionen ist hier nicht zu empfehlen!)

b) An das Punktmatrixdisplay ist ein Zeichengenerator angeschlossen, der das Wort 'FROHE OSTERN' mit einem anschließenden Leerzeichen ausgeben soll, indem der Generator jedes einzelne Zeichen der Zeichenkette zur Punktmatrixsteuerung überträgt. Zur Verfügung stehen Ihnen ein PLA und ein 7 Bit breites Register. Realisieren Sie den Generator mit Hilfe der beiden Bauteile. Gehen Sie davon aus, dass der Generator beim Start initial die Bitfolge '0000000' in seinem Register gespeichert hat.
(Tipp: Auch hier ist Minimieren von Funktionen nicht zu empfehlen!)

Aufgabe 5: PLA's

Mit PLA's lassen sich Schaltnetze einfach realisieren durch die Angabe des Bausteintyps innerhalb der PLA-Matrix. In dieser Aufgabe sind PLA's von verschiedene Bauteilen angegeben. Geben Sie für jedes dieser PLA an, welche Funktion realisiert wird bzw. um welches Bauteil es sich handelt.

a)

x_0	2
x_1	2
	1 f_0

b)

x_0	2	0
x_1	0	2
	1	1 f_0

c)

x_0	3	3	2
x_1	3	2	3
	1	1	1 f_0

d)

x_0	3	2
x_1	2	3
	1	1 f_0

e)

x_0	3
	1 f_0

f)

x_0	3
x_1	3
	1 f_0

g)

x_0	2	2	2	2	
x_1	3	2	3	2	
x_2	3	3	2	2	
	1	0	0	0	f_0
	0	1	0	0	f_1
	0	0	1	0	f_2
	0	0	0	1	f_3

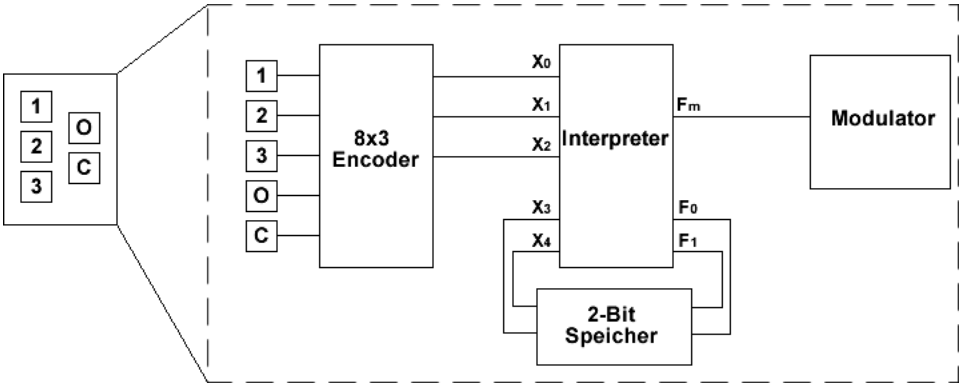
h)

x_0	2	0	0	0	
x_1	0	2	0	0	
x_2	0	0	2	0	
x_3	0	0	0	2	
	0	1	0	1	b_0
	0	0	1	1	b_1

i)

x_0	2	2	2	2	
x_1	3	2	3	2	
x_2	3	3	2	2	
	1	0	0	0	f_0
	0	1	0	0	f_1
	0	0	1	0	f_2
	0	0	0	1	f_3

Aufgabe 6: (⋆)Garagentor-Fernbedienung



In dieser Aufgabe sollen Sie einen Teil einer Garagentor-Fernbedienung entwerfen. Die Fernbedienung hat folgende Funktionen:
Tor 1 öffnen/schließen, Tor 2 öffnen/schließen, Tor 3 öffnen/schließen, alle Tore öffnen (O), alle Tore schließen (C). Sie sollen dabei konkret den Interpreter-Baustein realisieren. Die Fernbedienung funktioniert wie folgt:
Die Tasten der Fernbedienung sind an einen 8x3 Encoder angeschlossen, der die jeweilige Taste in eine Binärzahl umwandelt. Taste 1 ist an Eingang 0 des Encoders angeschlossen, Taste 2 an Eingang 1, Taste 3 an Eingang 2 usw.. Dabei ist zu bedenken, dass nicht alle Eingänge des Encoders benutzt werden und somit nicht alle Binärzahlen am Ausgang anliegen (z.B. wird das Wort $(101)_2$ nie am Ausgang ausgegeben, da es eine sechste Taste an der Fernbedienung nicht gibt).

Die 3 Ausgänge des Encoders sind mit dem Interpreter-Baustein verbunden (x_2,x_1,x_0). Der Ausgang F_m des Interpreter-Bausteins geht direkt zum Modulator, welcher vom Interpreterbaustein ein 4-Bit breites Datenwort seriell zugeschickt bekommt und das anschließend in ein Funksignal umwandelt. Dabei wird das LSB immer zuerst gesendet.
Die Eingänge x_3 und x_4 dienen dazu, in Kombination mit den Eingängen x_0, x_1 und x_2, das aktuell zu sendende Bit, das an F_m anliegen soll, zu bestimmen. Das Wort $(x_4, x_3)_2$ wird dabei inkrementiert und direkt an den Ausgang $(F_1, F_0)_2$ weitergeleitet. Die Ausgänge F_0 und F_1 werden anschließend in ein 2-Bit Register gespeichert und stehen im nächsten Takt wieder als x_3 und x_4 zur Verfügung, damit bestimmt werden kann, welches Bit als nächstes an F_m anliegt.

Beispiel (siehe hierzu Tabelle unten): Wenn Taste 3 gedrückt wird($(x_2, x_1, x_0)_2 = (011)_2$), soll die Zahl $(14)_{10} = (1110)_2$ an den Modulator seriell übertragen werden.
x_3 und x_4 sind anfangs beide 0. Es muss also das $(x_4, x_3)_2 = (00)_2 = (0)_{10}$ -te Bit übertragen werden - eine 0. An F_m liegt deswegen eine 0 an. Das Wort $(x_4, x_3)_2$ wird inkrementiert.

Im nächsten Takt ist $(x_4, x_3)_2 = (01)_2$. Es soll also das $(x_4, x_3)_2 = (01)_2 = (1)_{10}$ -te Bit übertragen werden - eine 1. An F_m liegt deswegen eine 1 an. Das Wort $(x_4, x_3)_2$ wird wieder inkrementiert usw.

Wenn $(x_4, x_3)_2 = (11)_2$ ist und inkrementiert wird, wird $(x_4, x_3)_2$ wieder auf $(00)_2$ gesetzt und der Vorgang beginnt von Neuem. Insgesamt werden nacheinander die Bits 0 (LSB), 1, 1,

1 (MSB) an den Modulator gesendet.
Die folgende Tabelle zeigt für jede Taste an, welcher Wert an den Modulator gesendet werden soll:

Taste	x_4	x_3	x_2	x_1	x_0	F_1	F_0	F_m	Zu sendender Dezimalwert
1									6
2									2
3	0	0	0	1	0	0	1	0	14
	0	1	0	1	0	1	0	1	
	1	0	0	1	0	1	1	1	
	1	1	0	1	0	0	0	1	
O									9
C									7

a) Füllen Sie die obige Tabelle aus.

b) Konstruieren Sie den Interpreter-Baustein mit Hilfe eines PLA.

Aufgabe 7: ($\star$)Einführung der MMIX-Umgebung

Diese Aufgabe soll in die Benutzung der MMIX-Umgebung einführen. Für weitere Informationen (Datenblätter, MMIX-Befehle, etc.) schauen Sie bitte auf folgende Webseite:

`http://www-kbsg.informatik.rwth-aachen.de/fileskbsg/legacy/teaching/SS2004/RS/MMIX_info/index.html`

Erstellen eines MMIX-Programms:

1. Quellcode erstellen und mit der Endung .mms speichern
2. Assemblieren durch: mmixal name.mms
3. Objektdatei ausführen mit: mmix name.mmo

a) Erzeugen Sie die Datei `einfach.mms`, die folgendes Programm enthält (Zeilennummern und Kommentare nicht mit eingeben):

```
 1          LOC #100          Start des Programms im Speicher
 2 x        IS $42            Anweisung, 'x' im Programm durch $42
                              zu ersetzen
 3 y        IS $43            ...
 4 z        IS $44            ...
 5 Main     SET x,40          Main: hier beginnt das Hauptprogramm
                              Setze x (also das Register 42, $42)
                              auf den Wert 40
 6          SET y,7           Setze y auf 7
 7          MUL $44,$42,$43   Multipliziere Reg. 42 mit Reg. 43,
                              schreibe das Ergebnis nach Reg. 44
 8          ADD z,z,3         ...
 9          DIV y,z,13        ...
10          TRAP 0,Halt,0     Ende des Programms
```

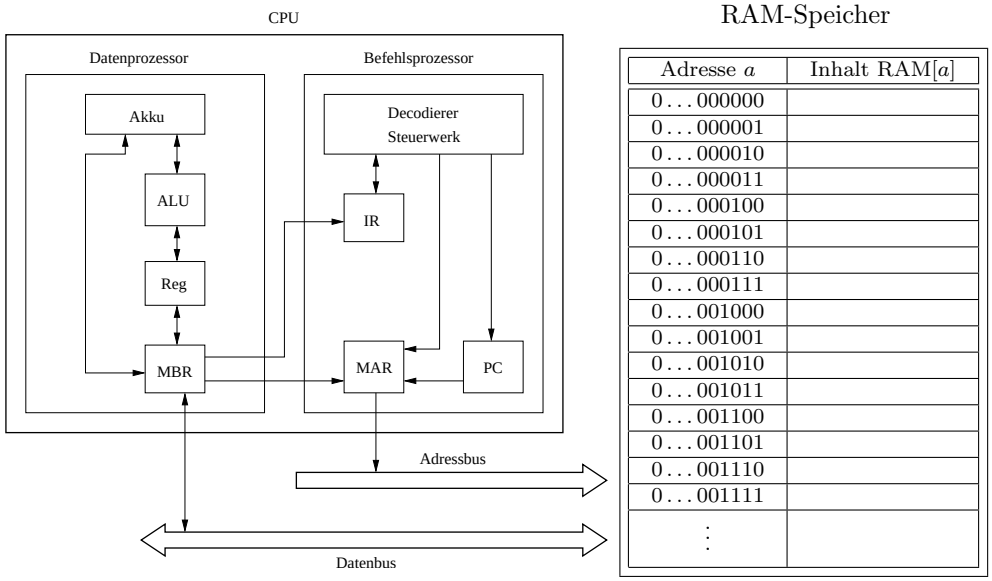
b) Erzeugen Sie den assemblierten Code `einfach.mmo`.

c) Welchen Wert enthalten die Register 42, 43 und 44 nach der Ausführung von Zeile 9 durch MMIX? Wie kommt man schnell an diese Information mit Hilfe von `mmix`?

d) Ändern Sie das Programm so ab, dass anfangs $x = 6$, $y = 7$ und $z = 20$ gilt, der Term $xy(x + y)/(z - y)$ berechnet wird und das Ergebnis in Register 14 steht.

Aufgabe 8: ($\star$) Arbeitsweise einer CPU

Die folgende Abbildung zeigt eine CPU-Struktur und ihre Verbindung zum RAM-Speicher. Im Vergleich zu der in der Vorlesung gezeigten Struktur einer CPU (VL 17 Folie 8), besitzt diese CPU ein zusätzliches Register Reg. Dafür wird hier zur Vereinfachung der genaue Aufbau des Registers Akku vernachlässigt: statt MR, L und A wird einfach nur Akku verwendet.



Für eine Speicheradresse a bezeichne RAM[a] die zugehörige Speicherzelle bzw. deren Inhalt. Nehmen Sie an, dass diesem Rechner die in der folgenden Tabelle durch ihre Operationscodes identifizierten Befehle/Operationen zur Verfügung stehen:

Operationscode	Wirkung(en) der Operation	
0	$(keineOperation)$	$\text{PC} \leftarrow \text{PC} + 1$
1	$\text{RAM}[\text{RAM}[\text{PC} + 1]] \leftarrow \text{Reg}$	$\text{PC} \leftarrow \text{PC} + 2$
2	$\text{Reg} \leftarrow \text{RAM}[\text{RAM}[\text{PC} + 1]]$	$\text{PC} \leftarrow \text{PC} + 2$
3	$\text{Reg} \leftarrow \text{RAM}[\text{PC} + 1]$	$\text{PC} \leftarrow \text{PC} + 2$
4	$\text{Reg} \leftarrow \text{Akku}$	$\text{PC} \leftarrow \text{PC} + 1$
5	$\text{Akku} \leftarrow \text{Reg}$	$\text{PC} \leftarrow \text{PC} + 1$
6	$\text{Akku} \leftarrow \text{Akku} + \text{Reg}$	$\text{PC} \leftarrow \text{PC} + 1$
7	$\text{Akku} \leftarrow \text{Akku} - \text{Reg}$	$\text{PC} \leftarrow \text{PC} + 1$
8	$\text{Akku} \leftarrow \text{Akku} \times \text{Reg}$	$\text{PC} \leftarrow \text{PC} + 1$
9	$\text{Akku} \leftarrow \text{Akku} \div \text{Reg}$	$\text{PC} \leftarrow \text{PC} + 1$
10	$(unbedingterSprung)$	$\text{PC} \leftarrow \text{RAM}[\text{PC} + 1]$
11	$\begin{cases} \text{Falls Akku} = 0\text{:} \\ \text{Falls Akku} \neq 0\text{:} \end{cases}$	$\begin{matrix} \text{PC} \leftarrow \text{RAM}[\text{PC} + 1] \\ \text{PC} \leftarrow \text{PC} + 2 \end{matrix}$

Der Rechner kenne nur ganze Zahlen (keine Gleikomma-Zahlen). Dementsprechend steht $\div$ für die ganzzahlige Division (z. B. $26 \div 7 = 3$).

a) Beschreiben Sie, wozu die Operationen mit den Operationscodes 1, 2, 3, 10, 11 dienen.

Im folgenden „Speicher-Auszug“ sind neben den Dualzahlen (mit denen „echte“ Rechner arbeiten) auch die entsprechenden Dezimalzahlen (klein und in Klammern) geschrieben.

In Ihren Lösungen brauchen Sie aber nur Dezimalzahlen anzugeben.

Es sei angenommen, dass die CPU hier bei jeder Befehlsverarbeitung die folgenden „Zuweisungen“ (in dieser Reihenfolge) durchführt:

MAR ← PC
MBR ← RAM[MAR]
IR ← MBR
Wirkung(en) der Operation IR
(PC-Änderung zuletzt!)

b) Protokollieren Sie den Inhalt der Register Akku, Reg, IR und PC nach jeder Befehlsverarbeitung bis PC den Wert 25 enthält, wobei zu Anfang alle Register 0 enthalten und der Speicher wie nebenstehend gefüllt ist.

Hinweis: PC sollte den Wert 25 nach 14 Befehlsverarbeitungen enthalten.

c) Betrachten Sie nun den nebenstehenden Speicherinhalt von Adresse 0 bis Adresse 21 (einschließlich) und interpretieren Sie ihn als ein Programm. Was wird durch dieses Programm berechnet?

Adresse a	Inhalt RAM[a]
(0) 0...000000	0...00000010 (2)
(1) 0...000001	0...00010110 (22)
(2) 0...000010	0...00000101 (5)
(3) 0...000011	0...00000010 (2)
(4) 0...000100	0...00010111 (23)
(5) 0...000101	0...00000111 (7)
(6) 0...000110	0...00000100 (4)
(7) 0...000111	0...00000001 (1)
(8) 0...001000	0...00011000 (24)
(9) 0...001001	0...00001011 (11)
(10) 0...001010	0...00011001 (25)
(11) 0...001011	0...00000011 (3)
(12) 0...001100	0...01100100 (100)
(13) 0...001101	0...00000101 (5)
(14) 0...001110	0...00000010 (2)
(15) 0...001111	0...00011000 (24)
(16) 0...010000	0...00001001 (9)
(17) 0...010001	0...00000100 (4)
(18) 0...010010	0...00000001 (1)
(19) 0...010011	0...00011000 (24)
(20) 0...010100	0...00001010 (10)
(21) 0...010101	0...00011001 (25)
(22) 0...010110	0...00001011 (11)
(23) 0...010111	0...00000100 (4)
(24) 0...011000	0...00000000 (0)
⋮	⋮

Sie kennen vermutlich schon die folgende Anekdote aus dem Leben von C. F. Gauß (bzw. eine Variante davon):

> Um seine Schüler (bzw. Gauß) eine Zeitlang zu beschäftigen (bzw. zu bestrafen), gab der Lehrer ihnen (bzw. ihm) die Aufgabe, die Zahlen von 1 bis 100 zu addieren. Doch der kleine Gauß hatte die Lösung 5050 schon nach kurzer Zeit gefunden.
>
> Gauß hatte nämlich erkannt, dass man statt $1 + 2 + 3 + \cdots + 98 + 99 + 100$ zu rechnen auch 50 „Paare“ $1 + 100$, $2 + 99$, $3 + 98$, ..., $50 + 51$ bilden kann, die jeweils 101 ergeben, und sich so die Lösung $50 \cdot 101 = 5050$ leicht berechnen lässt.

D. h.: Gauß benutzte die (nach ihm benannte) Gaußsche Summenformel $\sum_{k=1}^{n} k = \frac{1}{2}n(n+1)$.

Schreiben Sie je ein Programm, das die Summe s der ersten n natürlichen Zahlen berechnet ...

d) ... durch *fortgesetzte Addition* (also nach der Formel $s = \sum_{k=1}^{n} k$);

e) ... in sog. *geschlossener Form* (z. B. nach der Formel $s = \frac{1}{2}n(n+1)$).

Das Ergebnis s soll bei Adresse 3 gespeichert werden, wobei n bei Adresse 2 gespeichert sei und das Programm selbst bei Adresse 4 beginnt. Sie können davon ausgehen, dass $n \geq 0$ gilt. Kommentieren Sie Ihr Programm jeweils unbedingt ausführlich!

Adresse a	Inhalt RAM[a]
(0) 0...000000	0...00001010 (10)
(1) 0...000001	0...00000100 (4)
(2) 0...000010	*...******** (*n*)
(3) 0...000011	*...******** (*s*)
⋮	⋮

Hinweis zu Aufgabenteil d):
Beachten Sie, dass die Summationsreihenfolge durch $\sum_{k=1}^{n} k$ nicht festgelegt ist.[1]

Hinweis zu Aufgabenteil e):
Beachten Sie, dass sich $\frac{1}{2}n(n+1)$ auf verschiedene Arten berechnen lässt.[1]

[1] Nutzen Sie die Assoziativität, Kommutativität und ggf. Distributivität der Rechenoperationen aus, um möglichst einfache und schnelle (aber natürlich dennoch korrekte) Programme zu schreiben.