I. Übungen zu Band I

I.1 Grundlagen

Aufgabe 1: Zu den Maßeinheiten Kilo, Mega, Giga, ... (I.1.1)

a) Geben Sie Konstanten c und d zur Umrechnung von Zweier-Potenzen 2^x in 10er-Potenzen 10^y und umgekehrt an, für die gilt: $2^x = 10^{c \cdot x} = 10^y$ bzw. $2^x = 2^{d \cdot y} = 10^y$.

b) Tragen Sie in die untenstehende Tabelle die Umrechnungswerte für die gebräuchlichsten Maßeinheiten kilo, Mega, Giga und Tera zur Bezeichnung von Speicherkapazitäten (2^n byte/bit) und Übertragungsraten (10^n bit/s bzw. byte/s) – auf drei Nachkommastellen genau – ein. Geben Sie außerdem den absoluten Fehler f an, den man macht, wenn man die Unterschiede zwischen beiden Bedeutungen nicht berücksichtigt. Berechnen Sie auch den absoluten relativen Fehler r in Prozent (auf zwei Nachkommastellen genau), der auftritt, wenn man in Angaben von Übertragungsraten die genannten Maßeinheiten falsch interpretiert.

	2^x	$10^{c \cdot x}$	10^y	$2^{d \cdot y}$	f	r (%)
kilo	2^{10}	$10^{\text{-------}}$	10^3	$2^{\text{-------}}$		
Mega	2^{20}	$10^{\text{-------}}$	10^6	$2^{\text{-------}}$		
Giga	2^{30}	$10^{\text{-------}}$	10^9	$2^{\text{-------}}$		
Tera	2^{40}	$10^{\text{-------}}$	10^{12}	$2^{\text{-------}}$		

(Lösung auf Seite 150)

Aufgabe 2: Zu den Begriffen bit, byte, bit/s, ... (I.1.1)

a) Ein Speicherbereich sei 2^n byte groß. Er werde über eine bitserielle Leitung mit 10^m bit/s übertragen.

Geben Sie eine allgemeine Formel für die Übertragungsdauer T des Speicherbereichs in Sekunden [s] an.

b) Berechnen Sie die Übertragungsdauer für einen 512 kbyte großen Speicherbereich über eine bitserielle Leitung mit einer Übertragungsrate von 10 Mbit/s.

c) Welche Übertragungsdauer T' erhält man, wenn man fälschlicherweise die in b) vorgegebene Übertragungsrate als $10\ \text{Mbit/s} = 10 \cdot 2^{20}$ bit/s interpretiert? Wie groß ist der absolute relative Fehler r (in Prozent)?

(Lösung auf Seite 150)

Aufgabe 3: Moore'sches Gesetz (I.1.2)

Gordon Moore hatte im Jahr 1965 vorhergesagt, daß – unter einschränkenden, meist nicht erwähnten Bedingungen – die Anzahl der Transistoren auf einem Halbleiterchip in etwa jedes Jahr verdoppelt werden kann. Im Unterabschnitt I.1.2.2 wurde das daraus hergeleitete sog. Moore'sche Gesetz beschrieben, in dem häufig eine Verdopplung nach jeweils 1,5 Jahren unterstellt wird. Die im Bild 1.2-3 im Unterabschnitt I.1.2.2 eingezeichnete Gerade setzt bei Mikroprozessoren bereits einen größeren Zeitraum für die Verdopplung der Transistoren voraus.

a) Berechnen Sie die „realisierte" Anzahl von Jahren (auf drei Nachkommastellen), nach der jeweils eine Verdopplung der Transistorenzahl erreicht wurde, wenn man einen gleichmäßigen Verlauf der Entwicklung unterstellt. Gehen Sie dazu von den beiden folgenden Prozessoren aus:

   - 1971: Intel 4004, 2.250 Transistoren,
   - 2004: Intel Pentium 4E Prescott, 125 Millionen Transistoren.

   (Zur Kontrolle können Sie die Vorgaben für den Intel Pentium 4E im Diagramm in Bild 1.2-3 von Unterabschnitt I.1.2.2 nachtragen.)

b) Betrachten Sie den Fehler, den man macht, wenn man das Ergebnis aus a) „großzügig" zu einer der gebräuchlichen Zeitangaben „ein Jahr", „anderthalb Jahre", „zwei Jahre" rundet.

c) Berechnen Sie die maximale Anzahl der Transistoren pro Prozessorchip, die heute erreichbar sein müßten, wenn man von der Annahme des oft zitierten Moore'schen Gesetzes („Verdopplung alle 1,5 Jahre") oder sogar von Moore selbst („Verdopplung jedes Jahr") ausgeht.

(Lösung auf Seite 151)

Aufgabe 4: Leistungsangabe in MIPS (I.1.2)

Der Intel-Prozessor Pentium 4 kann im (unrealistischen) Idealfall pro Taktzyklus mit der Bearbeitung von jeweils drei x86-Befehlen bzw. drei daraus erzeugten RISC-ähnlichen Operationen beginnen.

a) Berechnen Sie die Leistung in MIPS *(Million Instructions per Second)* des Pentium 4 mit einer Arbeitstaktrate von 3,4 GHz.

b) Überprüfen Sie, ob dieser Leistungswert im Jahr 2003 noch dem Joy'schen Gesetz genügt bzw. geben Sie an, wie weit er von diesem entfernt ist.

(Lösung auf Seite 152)

I.2 Komponenten eines Mikroprozessors

Aufgabe 5: Mikroprogramm-Steuerwerk (Ampelsteuerung) (I.2.1)

Das folgende Bild 1 zeigt ein Mikroprogramm-Steuerwerk zur Steuerung einer Verkehrsampel. Es wird durch einen 1-Hz-Takt betrieben, der durch eine „Vorteilerschaltung 1:10“ *(Prescaler)* auf eine Frequenz von 0,1 Hz verlangsamt werden kann. Die Aktivierung der Vorteilerschaltung geschieht durch ein Eingangssignal (EN – *Enable*), das im L-Pegel aktiv ist.

Die Ampelanlage unterstütze zwei Betriebsmodi:

- Im „Normalbetrieb“ durchlaufe sie zyklisch die übliche Zustandsfolge: Rot, Rot-Gelb, Grün, Gelb. Dabei sollen die Zustände Rot und Grün jeweils 20 Sekunden, die Zwischenzustände Rot-Gelb und Gelb jedoch nur 10 Sekunden dauern.
- Im „Störbetrieb“ soll nur das gelbe Licht (G) im 2-Sekunden-Takt blinken, also jeweils 1 Sekunde an und 1 Sekunde aus sein. Die beiden anderen Lampen (R, Gr) sollen dabei ausgeschaltet sein.

Die Auswahl zwischen beiden Modi geschehe durch ein Signal S, das (durch einen ‚1‘-Zustand) das Vorliegen einer Störung anzeigen soll.

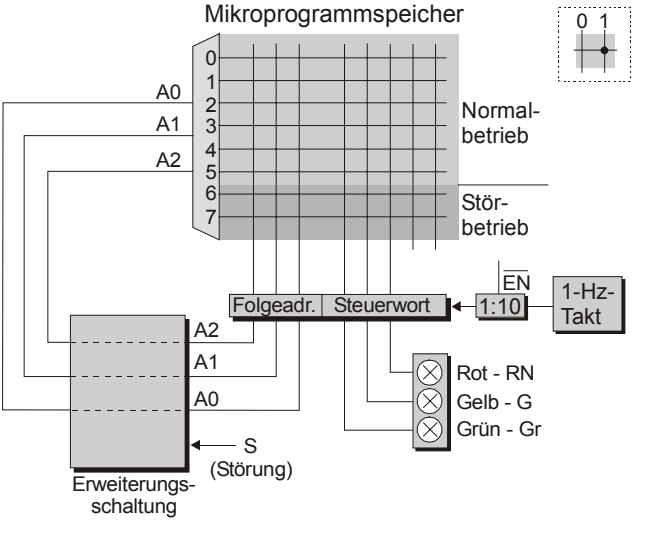


Bild 1: Mikroprogramm-Steuerwerk

a) Tragen Sie das Mikroprogramm für die Zustandsfolge im Normalbetrieb in den Mikroprogrammspeicher ein. Kennzeichnen Sie dazu eine logische ‚1' deutlich durch einen Knoten, wie es im Kasten rechts oben im Bild dargestellt ist. Für das Mikroprogramm stehen Ihnen die sechs ersten Speicherwörter (0 – 5) zur Verfügung.

   Geben Sie eine kurze Beschreibung Ihres Mikroprogramms an, in der Sie auf die Programmierung der Folgeadressen, der Steuerwörter und der Ansteuerung des *Enable*-Eingangs EN des Frequenzteilers eingehen.

b) Tragen Sie in den Speicherwörtern 6 und 7 zwei Mikrobefehle ein, die den Blinkmodus im Störbetrieb realisieren. Erweitern Sie das Bild durch eine Schaltung, die dafür sorgt, daß in diesem Modus auf den Adreßleitungen zyklisch die Adressen 6 und 7 ausgegeben werden, solange das Störsignal S = 1 ist. Dabei muß sichergestellt werden, daß nach Verlassen des Blinkmodus (S $\rightarrow$ 0) zunächst der Zustand Rot eingenommen wird.

   Geben Sie auch hier eine kurze Beschreibung Ihrer Mikrobefehle an, in der Sie auf die Programmierung der Folgeadressen, der Steuerwörter und der Ansteuerung des *Enable*-Eingangs EN des Frequenzteilers eingehen. Beschreiben Sie auch die Funktion Ihrer Erweiterungsschaltung!

(Lösung auf Seite 152)

Aufgabe 6: Mikroprogramm-Steuerwerk (Dualzähler) (I.2.1)

Geben Sie das Schaltbild eines einfachen Mikroprogramm-Steuerwerks an, das – unter anderen Operationen – einen 2-bit-Aufwärts/Abwärts-Zähler realisiert. Die Zählrichtung werde durch ein Signal U#/D *(up/down)* vorgegeben. Der Zählvorgang werde durch einen Maschinenbefehl COUNT (im Befehlsregister) angestoßen, der durch den Befehlsdecoder auf die binäre Unteradresse …10 000_2 des Mikroprogrammspeichers abgebildet wird. Die Taktsteuerung des Mikroprogramm-Steuerwerks werde durch ein einzelnes Bit im Mikrobefehl vorgenommen. Stellen Sie die Zustandsfolge der Zählerbits für beide Betriebsmodi dar.

(Lösung auf Seite 153)

Aufgabe 7: Mikroprozessor-Signale (HALT, HOLD) (I.2.2)

Im Unterabschnitt I.2.2.2 wurden die beiden Signale HALT und HOLD beschrieben, durch die ein Prozessor in einen „Haltezustand" versetzt werden kann. Geben Sie die wesentlichen Unterschiede zwischen diesen beiden Prozessor-Signalen an.

(Lösung auf Seite 154)

Aufgabe 8: Interruptsteuerung (I.2.2)

Auf einem Computersystem mit einer Taktfrequenz von 1 MHz läuft in einer Endlosschleife ein Programm ab, während dessen Ausführung das IE-Flag *(Interrupt Enable)* ständig gesetzt ist, Unterbrechungen also zugelassen sind. Für eine Systemuhr wird alle 100 Mikrosekunden eine nicht maskierbare Unterbrechungsanforderung (NMI) erzeugt, deren Ausführungsroutine 20 Taktzyklen benötigt. Der NMI#-Eingang wird dazu 10 Zyklen lang auf L-Potential gezogen; das sei länger, als die Ausführung jedes Maschinenbefehls (max. 7 Zyklen) dauert. Außerdem erzeugt ein angeschlossenes Gerät in unregelmäßigen Abständen immer dann eine maskierbare Unterbrechungsanforderung (IRQ), wenn es ein Zeichen an den Prozessor zu übertragen hat. Die IRQ-Routine dauert 40 Taktzyklen, der IRQ#-Eingang wird für 30 Zyklen auf L-Pegel gehalten. Das Diagramm in Bild 2 zeigt die auftretenden Unterbrechungsanforderungen.

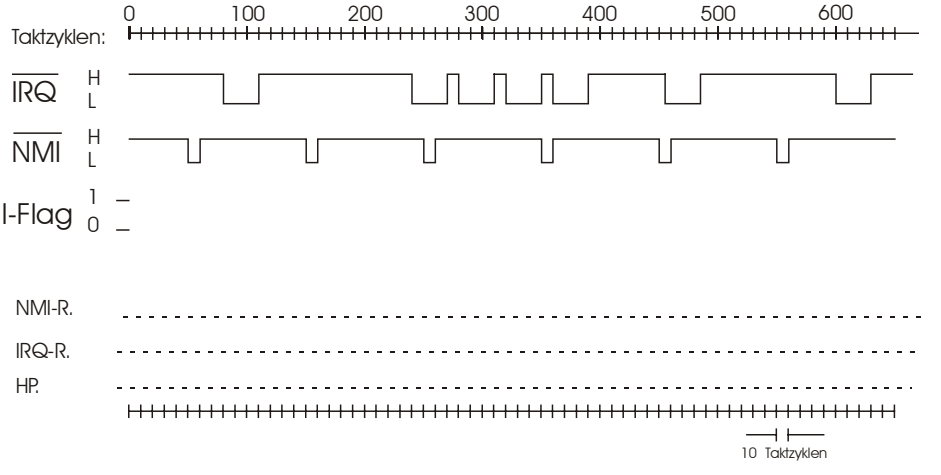


Bild 2: Zeitliche Lage der Unterbrechungsanforderungen

Vervollständigen Sie das Diagramm, indem Sie den Zustand des IE-Flags einzeichnen. Tragen Sie über der unteren Zeitachse in Form einer Treppenfunktion für jeden Zyklus ein, welchen Programmteil (Hauptprogramm, IRQ- oder NMI-Routine) der Prozessor gerade bearbeitet. Kennzeichen Sie dabei alle Zeitpunkte, zu denen eine neue IRQ- oder NMI-Routine gestartet wird. Die Reaktionszeit zum Umschalten auf eine Unterbrechungsroutine darf vernachlässigt werden, d.h. es wird angenommen, daß jede Anforderung genau zwischen zwei Befehlsausführungen auftritt.
Werden alle IRQ-Anforderungen ausgeführt?

(Lösung auf Seite 155)

Aufgabe 9: Interruptschachtelung (I.2.2)

a) Ein Mikroprozessor besitze einen jeweils 16 bit breiten Adreß- und Datenbus. Sein integrierter Interrupt-Controller verwalte bis zu 7 Interruptquellen, die sich über eine Interruptvektor-Nummer IVN $\in \{1, 2, ..., 7\}$ identifizieren. (Die IVN = 0 kennzeichnet den Zustand „keine Interruptanforderung".) Der Beginn der Interruptvektor-Tabelle werde durch ein 16-bit-Register IBR festgelegt. Im Arbeitsspeicher liege die folgende Interruptvektor-Tabelle 1, deren Einträge im *Little-Endian*-Format aufzufassen sind.

Tabelle 1: Interruptvektor-Tabelle

Byte Adresse	0	1	2	3	4	5	6	7
A780	80	B0	76	70	80	B0	04	C0
A788	A0	C0	66	50	44	20	7C	99

Geben Sie die Belegung des IBR für die oben stehende Interruptvektor-Tabelle an:

IBR = $……...

Tragen Sie in die folgende Tabelle für alle Interruptvektor-Nummern die Startadresse der *Interrupt Service Routine* (ISR) ein.

IVN	Startadresse	Priorität
1		
2		
3		
4		
5		
6		
7		

Geben Sie die Formel an, nach der aus der IVN und dem Inhalt des Basisregisters IBR die Startadresse der zugehörigen *Interrupt Service Routine* (ISR) berechnet wird:

Startadresse ISR = ……..

Die Abarbeitung gleichzeitig anliegender Unterbrechungswünsche werde vom Interrupt-Controller durch Prioritäten gesteuert, wobei eine höherwertige IVN eine höhere Priorität bedeute.

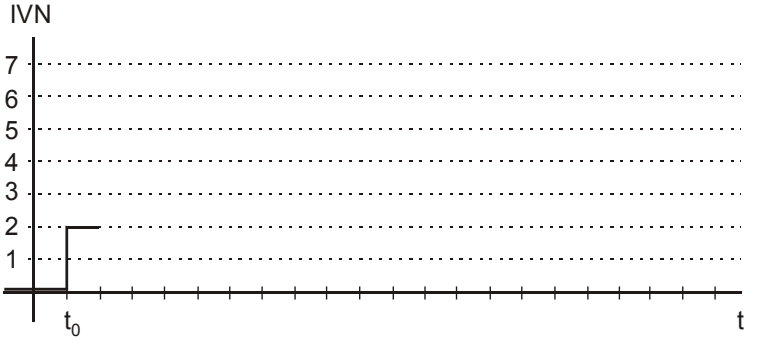
Zeichnen Sie in der mit „Priorität" bezeichneten Spalte der oben stehende Tabelle einen Pfeil von der niedrigsten zur höchsten Priorität!

b) Der Interrupt-Controller des Mikroprozessors unterstütze die Schachtelung von Interrupts, d.h. er erlaube die Unterbrechung einer *Interrupt Service Routine* ISR1 durch eine andere Routine ISR2. Diese Routine muß jedoch eine höhere Priorität besitzen, d.h. für ihre Interruptvektor-Nummern muß gelten: IVN1 > IVN2.

Ihre Aufgabe ist es nun, die Ausführung einer Reihe von Unterbrechungsanforderungen zu untersuchen:

- Zu Beginn des Beobachtungszeitraums t_0 werde eine ISR mit IVN = 2 ausgeführt.
- Während ihrer Bearbeitung trete ein Interrupt der Priorität 3 auf.
- Während der Bearbeitung dieses Interrupts werden weitere Interrupts der Prioritäten 1 und 4 sowie erneut der Priorität 3 angefordert.
- Während der Bearbeitung des zweiten Interrupts der Priorität 3 soll der Reihe nach
  - ein Interrupt der Priorität 7 und während dessen Bearbeitung ein Interrupt der Priorität 5 auftreten,
  - nach Beendigung des Interrupts mit der Priorität 5 ein Interrupt mit der Priorität 2 erfolgen.
- Nach Abarbeitung aller ISR soll das Hauptprogramm (mit der Priorität 0) fortgesetzt werden.

Tragen Sie in das folgende Diagramm alle Übergänge zwischen den Interrupt-Service-Routinen und ihre Verläufe ein. Gehen Sie vereinfachend davon aus, daß jede Ausführung einer ISR genau zwei Zeiteinheiten dauert und Übergänge nur zu den markierten Zeitpunkten geschehen können. Kennzeichnen Sie den Übergang von einer ISR zu einer ISR mit derselben IVN durch einen senkrechten Trennstrich.



c) Kann eine ISR mit der IVN = 7 unterbrochen werden? (Begründung!)
Welche Rolle übernimmt daher ein Interrupt mit der IVN = 7?

(Lösung auf Seite 156)

Aufgabe 10: Interruptsteuerung (Motorola MC680X0) (I.2.2)

Gegenstand dieser Aufgabe ist die Interruptsteuerung der Motorola-Prozessoren MC680X0, die im Exkurs I.2.2.7 beschrieben wurde. Durch das Betriebssystem werde die Interruptmaske im Steuerregister SR des MC680X0 zunächst auf den Wert 2 gesetzt. Nun trete ein Interrupt der Klasse 3 auf. Während der Bearbeitung dieses Interrupts werden weitere Interrupts der Klassen 1, 3 und 4 angefordert. Während der Bearbeitung des zweiten Interrupts der Klasse 3 soll der Reihe nach ein NMI sowie ein Interrupt der Klasse 5 auftreten.

Geben Sie alle Zustände der Maskenbits $I_2 - I_0$ an.

(Lösung auf Seite 157)

Aufgabe 11: Interruptvektor-Tabelle (Lage und Größe) (I.2.2)

Das 32-bit-Basisadreß-Register einer Interruptvektor-Tabelle IVT habe den Wert $A000 8C00. Die Tabelle enthalte 256 Einträge. Jeder Eintrag sei 4 byte lang. Geben Sie für die Interruptvektor-Nummer IVN = 22 die Speicheradressen, unter denen die Startadresse der zugehörigen Ausnahme-Behandlungsroutine ISR *(Interrupt Service Routine)* in der Tabelle abgelegt ist, sowie den Adreßbereich an, der von der Tabelle belegt wird.

Lage der Startadresse: ..... – .....

Lage der Vektortabelle: ..... – .....

Welche Information wird zusätzlich benötigt, um die Startadresse der ISR angeben zu können?

(Lösung auf Seite 157)

Aufgabe 12: Interruptvektor-Tabelle (Interrupt-Startadressen) (I.2.2)

Ein Prozessor besitze eine Interruptvektor-Tabelle IVT, deren Lage im Arbeitsspeicher über das Basisadreßregister *(Interrupt Vector Base Register)* BR festgelegt werden kann. Das Basisadreßregister habe den Wert BR = $30AB A000. Jeder Eintrag in dieser Tabelle stellt die Startadresse einer Interrupt-Behandlungsroutine ISR im *Little-Endian*-Format dar und wird über eine 8 bit lange Interruptvektor-Nummer IVN selektiert.

Die folgende Tabelle 2 zeigt drei Ausschnitte aus der Belegung des Arbeitsspeichers.

a) Geben Sie die Größe und Lage der IVT im Speicher an, wenn diese maximal groß ist:

   Größe: ..... byte, Anfangsadresse: $....., Endadresse: $.....

Tabelle 2: Ausschnitte aus der Belegung des Arbeitsspeichers

Adresse \ X	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0
F700 802X	00	6F	F9	FE	5D	43	AC	56	5F	EE	78	98	BC	76	5F	67
F700 801X	44	DE	FC	56	6F	ED	E3	21	FC	33	FF	37	AA	54	78	00
F700 800X	3A	CC	54	FF	ED	CC	52	AA	B6	B7	B8	54	43	32	DD	DE
30AB A03X	66	DC	DD	AD	DA	CC	54	F5	F7	00	80	14	FC	FC	CD	DC
30AB A02X	66	77	FD	DF	ED	4E	4D	D3	55	DC	AD	DE	EF	FF	65	DC
30AB A01X	2A	B0	7F	F8	55	66	DF	FD	FF	FC	CD	ED	DF	DE	55	3A
30AB A00X	CF	AA	55	00	11	FF	FC	CD	ED	DA	55	DE	DD	DA	AA	DC
2AB0 7FFX	55	66	DC	DE	AE	DE	4E	BD	DC	5E	4A	56	DE	DF	DD	EF
2AB0 7FEX	60	FF	F1	F4	EC	55	5F	78	BB	BA	FD	E3	D4	63	DF	77

b) Tragen Sie in die folgende Tabelle für die (dezimalen) Interruptvektor-Nummern IVN = 7 und IVN = 13 die
   - Startadresse des Interruptvektors in der IVT,
   - die Startadresse der Interrupt-Behandlungsroutine ISR,
   - das erste Byte des ersten Befehls der ISR ein.

IVN (dez.)	Startadresse des Int.-Vektors in IVT	Startadresse der ISR	1. OpCode-Byte der ISR
7			
13			

(Lösung auf Seite 158)

Aufgabe 13: Indizierte Adressierung (I.2.3)

Im Abschnitt I.3.3 haben Sie die folgende Schreibweise für eine indizierte Adressierungsart mit Post-Inkrementierung kennengelernt:

<Offset>(B0)(I0)+ .

Die dadurch vorgegebene Berechnungsvorschrift für die Operandenadresse lautet:
„*Addiere den Inhalt des (Basis-)Registers B0, des (Index-)Registers I0 sowie den im Befehl angegebenen Offset. Erhöhe nach dem Zugriff zum Operanden den Inhalt des (Index-)Registers I0 um 1*“.

Geben Sie alle Teilschritte an, die zur Ausführung der Adreßberechnung nötig sind. Nehmen Sie dabei an, daß die Inkrementierung des (Index-)Registers I0 durch das Adreßwerk vorgenommen werden muß.

(Lösung auf Seite 159)

Aufgabe 14: Virtuelle Speicherverwaltung (I.2.3)

Im folgenden Bild 3 wird ein Beispiel einer einfachen „virtuellen“ Speicherverwaltung dargestellt, deren Aufgabe die Umwandlung von logischen (virtuellen) 30-bit-Adressen in physikalische 24-bit-Adressen ist.

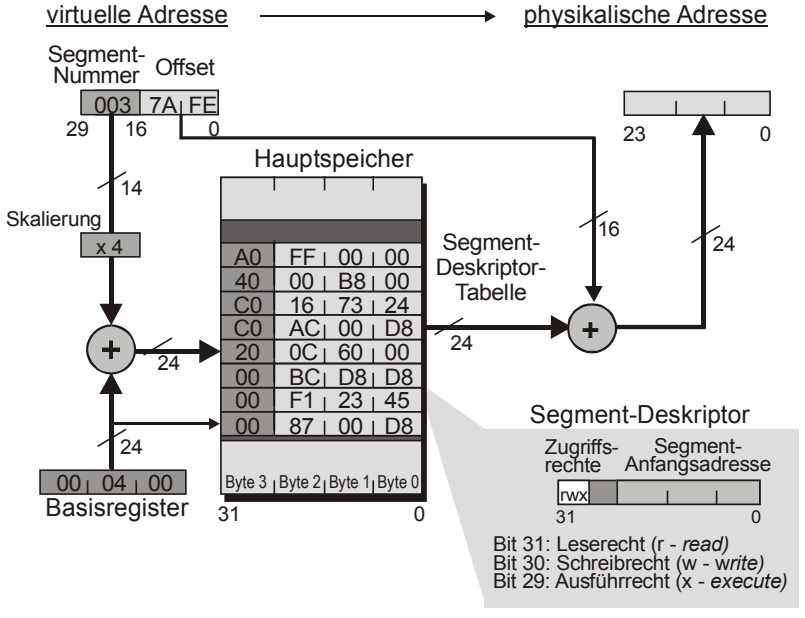


Bild 3: Prinzip der virtuellen Speicherverwaltung

Der (virtuelle) Programm-Adreßraum wird dazu in Bereiche („Segmente“) unterteilt, die (in unserem Beispiel – zunächst –) eine feste Länge von 64 kbyte haben. In einer Tabelle im Hauptspeicher, der sog. Segment-Deskriptor-Tabelle (SDT), befindet sich für jeden Adreßbereich ein 4-byte-Eintrag, der Segment-Deskriptor, der die 24-bit-Anfangsadresse des Bereiches im Hauptspeicher sowie gewisse „Zugriffsrechte“ zu diesem Bereich enthält. Diese Zugriffsrechte regeln, ob ein Programm auf dieses Segment lesend oder schreibend zugreifen oder es als Programmcode ausführen darf. (Ein gesetztes Bit erlaubt den jeweiligen Zugriff.) Bei einem Segmentzugriff ohne entsprechend gesetztes Zugriffsrecht wird eine Ausnahme *(Exception)* erzeugt. Sind die Bits 31 bis 29 sämtlich 0, so soll dies bedeuten, daß zu diesem Tabelleneintrag kein gültiges Segment im Hauptspeicher existiert. Falls der Zugriff auf die SDT indirekt über ein Basisregister geschieht, kann sie an einer beliebigen Stelle im Hauptspeicher angelegt werden.

Bei der Einlagerung von neuen Segmenten muß das Betriebssystem beachten, daß es kein Segment anlegt, welches sich mit einem bestehenden Segment überlappt. Falls eine Überlappung vorliegt, müssen die überdeckten Segmente ausgelagert werden.

Außerdem darf das Betriebssystem keinen Segment-Deskriptor überschreiben, dessen Segment durch seine Zugriffsrechte nicht als ungültig gekennzeichnet ist.

Wie im Bild graphisch dargestellt, geschieht der Zugriff auf einen Speicherwert durch die Angabe einer virtuellen (logischen) 30-bit-Adresse, die aus einer 14-bit-Segmentnummer und einem 16-bit-*Offset* besteht. Die Segmentnummer dient als Index zur Selektion eines bestimmten Segment-Deskriptors, der *Offset* bestimmt danach die Position des Wertes in diesem Segment.

a) Warum wird vor der Addition des Basisregister-Inhalts die Segmentnummer mit 4 skaliert (d.h. multipliziert)? Wie groß sind (im behandelten Beispiel) die maximalen logischen und physikalischen Adreßräume?

b) Im Bild 3 ist ein Ausschnitt der SDT im Hauptspeicher und der Inhalt des Basisregisters dargestellt. Welche physikalische Adresse wird mit der virtuellen Adresse $0037 AFE angesprochen und mit welchen Rechten darf auf diese physikalische Adresse zugegriffen werden?

c) Wie ändert sich die Deskriptor-Tabelle im Bild 3, wenn ein Segment mit der Nummer 6 und den Rechten zum Lesen, Schreiben und Ausführen ab der physikalischen Adresse $12A0F0 angelegt wird? Was muß das Betriebssystem beachten, wenn ein neues (64-kbyte-)Segment angelegt wird?

d) Wie viele Bytes des physikalischen Speichers stehen maximal für Programme und Daten zur Verfügung, wenn jedes Segment genau 64 kbyte lang ist, die SDT selbst maximal 64 kbyte groß ist und kein Ein-/Auslagerungsvorgang zwischen dem Hauptspeicher und einem Peripheriespeicher (z.B. der Festplatte) stattfinden soll?

e) Nun werde die Voraussetzung fester Segmentlängen fallengelassen, d.h. es werden Segmente variabler Länge bis zu 64 kbyte zugelassen. Skizzieren Sie den Aufbau eines möglichen Segment-Deskriptors, wenn seine Lage innerhalb der SDT weiterhin durch einfache Skalierung der Segmentnummer in der virtuellen Adresse mit einer Potenz von 2 gefunden werden soll.

   Mit welchem Wert muß diese Skalierung durchgeführt werden? Wie viele Segmente können nun im Hauptspeicher maximal verwaltet werden? Wie viele Bits werden für die Segmentnummer in der virtuellen Adresse benötigt?

(Lösung auf Seite 160)

Aufgabe 15: Arithmetisch/logische Einheit (ALU) (I.2.4)

Betrachtet werde eine 8-bit-ALU, die im wesentlichen aus einem Parallel-Addierer/Subtrahierer besteht. Durch ein Steuersignal AS werde angezeigt, ob die augenblicklich ausgeführte Operation eine Addition (AS = 0) oder eine Subtraktion (AS = 1) zweier Operanden im Zweierkomplement ist.

Die Operanden seien mit $A = A_7...A_0$ und $B = B_7...B_0$, das Ergebnis mit $F = F_7...F_0$ bezeichnet.

a) Leiten Sie logische Gleichungen zur Bestimmung der *Flags* AF, CF, OF, ZF, SF, EF, PF aus A, B, F und AS her.

b) Nun sei angenommen, daß die ALU ihre Operanden wahlweise als vorzeichenlose ganze Zahlen oder als vorzeichenbehaftete Zahlen im Zweierkomplement verarbeiten kann. Leiten Sie für beide Fälle aus den *Flags* logische Beziehungen dafür her, daß für die beiden Operanden A, B einer Vergleichssubtraktion A – B gilt:

$$A < B,\ A \le B,\ A > B,\ A \ge B.$$

c) Zeichnen Sie eine Schaltung, die aus den Operanden A, B, dem Ergebnis F sowie dem Steuersignal AS die Flags ZF, EF, SF, PF, OF erzeugt und aus diesen sowie aus dem Übertragsbit CF und dem Hilfs-Übertragsbit AF wiederum die unter b) hergeleiteten Vergleichsresultate. Ergänzen Sie Ihre Zeichnung um einen Testmultiplexer, der genau eines der aufgeführten Signale zum Steuerwerk durchschaltet.

d) A, B seien nun zweistellige BCD-Zahlen. Geben Sie an, wie die ALU nach einer Addition A + B das Ergebnis F wiederum in eine zweistellige BCD-Zahl verwandeln kann.

(Lösung auf Seite161)

Aufgabe 16: Bereichsüberschreitung der ALU (I.2.4)

a) Im Abschnitt I.2.4 haben Sie das *Overflow Flag* OF im Statusregister eines Mikroprozessors kennengelernt, das immer dann gesetzt wird, wenn im Zweierkomplement eine Bereichsüberschreitung auftritt.

   i. Geben Sie in allgemeiner Form an, wann bei der Addition eine Bereichsüberschreitung vorliegt.

   ii. Gegeben seien zwei vorzeichenbehaftete 8-bit-Zahlen A und B im Zweierkomplement. Tragen Sie in die folgende Tabelle die Ergebnisse der Addition ein und geben Sie jeweils an, ob nach der Operation das OF-Bit gesetzt ist oder nicht.

		A					
		\$A6	OF	S	\$6C	OF	S
B	\$7A						
	\$80						

   iii. Tragen Sie in die mit ‚S' beschrifteten Spalten der Tabelle unter ii. die Zahlen ein, die man erhält, wenn das betrachtete Rechenwerk mit „Sättigung" arbeitet.

b) Das Statusregister eines 16-bit-Prozessors enthalte ein Bit UOF *(Unsigned Overflow Flag)*, das genau dann gesetzt wird, wenn bei der Verknüpfung zweier vorzeichenloser ganzer Zahlen eine Bereichsüberschreitung stattfindet. Tragen Sie in die folgende Tabelle die Ergebnisse der Addition der angegebenen Zahlen und den Zustand des Bits UOF nach der Operation ein. Geben Sie in den

mit ‚S' bezeichneten Spalten wiederum die Ergebnisse an, die man erhält, wenn das Rechenwerk mit Sättigung arbeitet.

		A		
		$A604	UOF	S
B	$40A5			
	$8066			

c) Die 8-bit-Register R0, R1, R2 eines Mikroprozessors werden mit den folgenden Werten initialisiert: R0 := $1A, R1 := $80, R2 := $08.

Der Prozessor führe danach die nachstehende Operationsfolge solange aus, bis im Register R3 ein Wert ungleich $00 steht:

R3 := R0 ^ R1	*Logische Und-Verknüpfung*
R2 := R2 – 1	*Dekrementieren um 1*
R1 := >> R1	*Logisches Rechtsschieben um 1 bit*

Welche Werte stehen danach in R0 bis R3?
Welche Bedeutung hat der Wert in R2 nach Beendigung aller Operationen?

(Lösung auf Seite 165)

Aufgabe 17: Schiebe- und Rotationsoperationen (I.2.4)

Es seien A := 10101101_2 und CF := 0. Geben Sie für alle im Abschnitt I.2.4 beschriebenen Schiebe- und Rotationsoperationen das Ergebnis an.

(Lösung auf Seite 165)

Aufgabe 18: 4-bit-Multiplexer (I.2.4)

a) Geben Sie eine Realisierung für einen 4-bit-Multiplexer derart an, daß der Index des durchgeschalteten Eingangs E_i gerade mit der durch die Steuerleitungen S_1, S_0 gegebenen Dualzahl übereinstimmt.

b) Entwerfen Sie ein Schieberegister, das in Abhängigkeit von zwei Steuersignalen S_1, S_0 wahlweise die folgenden Funktionen ausführt:

S_1	S_0	Register ...
0	0	... unverändert lassen
0	1	... parallel über die Dateneingänge D_i laden
1	0	... mit dem Doppelten des alten Registerinhalts laden
1	1	... mit dem Vierfachen des alten Registerinhalts laden

c) Betrachten Sie Bild I.2.3-1. Geben Sie eine Hardwarelösung für die Beschaltung der Eingänge des Adreßaddierers an, um eine Skalierung der IVN um den konstanten Faktor 8 zu erreichen.

(Lösung auf Seite 166)

Aufgabe 19: MMX-Operationen (einfache Operationen) (I.2.4)

Die 64-bit-MMX-Register R0, R1 enthalten die folgenden hexadezimalen Werte:

R0 = (7FAD 84AE 59CA FCFF), R1 = (AD5F 01DD 6C00 F6FA).

Geben Sie die Werte der Register nach der Ausführung der folgenden MMX-Operationen an. Dabei werde in den Operationen durch die Buchstaben B, W, D, Q angegeben, ob die Registerinhalte als gepackte Bytes, Wörter, Doppelwörter oder Quadwords aufgefaßt werden sollen.

a) Logisches Linksschieben (D):

R0 = (.....)

b) Arithmetisches Rechtsschieben (W):

R1 = (.....)

c) Vorzeichenbehaftete Addition mit Sättigung (W):

R0 + R1 = (.....)

d) Vorzeichenlose Addition mit Sättigung (B):

R0 + R1 = (.....)

e) Vorzeichenloser Vergleich auf „größer als“ (W):

R0 > R1 = (.....)

(Lösung auf Seite 168)

Aufgabe 20: MMX-Operationen (Operationenfolge) (I.2.4)

Ein MMX-Rechenwerk stellt die parallele (vorzeichenlose) Vergleichsoperation > (größer als – *gt*) sowie die bitweise, parallel ausgeführten logischen Operationen $\wedge$ (und), $\vee$ (oder), $\neq$ (Antivalenz) zur Verfügung. Die ebenfalls bitweise ausgeführte logische Operation $-\wedge$ (Inhibition) invertiert zunächst den „linken“ Operanden und bildet dann die Und-Verknüpfung mit dem „rechten“ Operanden. Die Transportoperation := überträgt (u.a.) einen Operanden in ein MMX-Register.

Die Operationen (op) werden im Zweiadreß-Format auf den Inhalten der 64-bit-MMX-Register Rn ausgeführt: Ri := Ri op Rj. Das bedeutet, daß das erste Operandenregister Ri durch das Ergebnis der Operation überschrieben wird.

a) Geben Sie für die Anfangsbelegung der Register R0, R1 mit jeweils vier 16-bit-Wörtern (in Hexadezimaldarstellung):

R0 = (CD70 5F8C 0DC5 95F4), R1 = (B5F0 FFFF 105A 94FF)

die Ergebnisse der nachstehenden Operationen an, die parallel auf den vier Paaren von 16-bit-Wörtern in den Registern ausgeführt werden.

R2 := R0	R2 = (.....)
R2 := R0 > R1	R2 = (.....)
R3 := R2	R3 = (.....)
R3 := R3 $-\wedge$ R0	R3 = (.....)
R2 := R2 $\wedge$ R1	R1 = (.....)
R2 := R2 $\vee$ R3	R2 = (.....)

b) Was berechnet diese Operationsfolge?

c) Geben Sie eine Operationsfolge an, die parallel die Maxima über die 16-bit-Teildaten der Register R0 und R1 berechnet.

(Lösung auf Seite 168)

Aufgabe 21: Universelles Schiebe-/Zähl-Register (I.2.5)

Skizzieren Sie den Aufbau eines universellen Registers, das alle erwähnten Funktionen ausführen kann: Rücksetzen, Vorwärts/Rückwärts-Schieben, Vorwärts/Rückwärts-Zählen.

(Lösung auf Seite 168)

Aufgabe 22: Register mit automatischer Modifikation (I.2.5)

Ein 8-bit-Mikroprozessor besitze ein 16-bit-Basis(adreß)register BR sowie ein 8-bit-Indexregister IR. Bei der indizierten Adressierung mit 8-bit-Offset werde die effektive Adresse EA eines Operanden durch die Addition des Inhalts von IR und eines im Befehl angegebenen Offsets zum Inhalt von BR gewonnen. Die Inhalte von IR und BR werden dabei als vorzeichenlose ganze Zahlen, der Offset als vorzeichenbehaftete Zahl im Zweierkomplement aufgefaßt.

Die Adressierungsart und die Anfangsbelegung des Speichers sind im folgenden Bild 4 dargestellt.

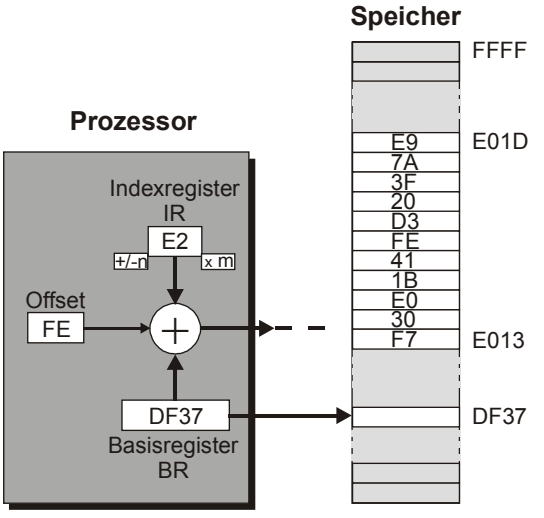


Bild 4: Indizierte Adressierung mit Offset

a) Das Indexregister IR verfüge über die Möglichkeit der automatischen Modifikation (autoinkrement/autodekrement) um die Werte n = 1, 2, 4.

Geben Sie in der unten stehenden Tabelle an, welche effektive Adresse EA jeweils durch einen Lesebefehl mit indizierter Adressierung mit Offset ausgegeben wird, welches Byte B dadurch im Speicher angesprochen wird und welchen Wert das Indexregister IR (nach der Befehlsausführung) hat, wenn n = 4 ist und alle Möglichkeiten der Modifikation betrachtet werden.

Im oben stehenden Bild sind dazu die aktuellen Werte für die Register BR, IR und den Offset angegeben. Diese Werte sollen jeweils für alle Teilaufgaben erneut zugrunde gelegt werden.

Modifikation	EA	B	IR
ohne			
predekrement			
postdekrement			
preinkrement			
postinkrement			

b) Das Indexregister IR besitze nun zusätzlich die Möglichkeit der Skalierung mit dem Faktor m = 1, 2, 4. Die aktuellen Inhalte der Register und der Wert des Offsets seien:

BR = \$DF37, IR = \$39, Offset = \$00.

Welche effektive Adresse EA wird durch einen Lesebefehl angesprochen, welches Byte B wird dadurch aus dem Speicher geholt und welchen Wert hat das Register IR nach der Ausführung der Operation, wenn m = 4, n = 1 sind und die Adressierung mit Predekrement-Modifikation des Indexregisters gewählt wird?

EA = ... = $....., B = $....., IR = $.....

c) Die indizierte Adressierung mit Offset aus a) werde nun indirekt angewandt, d.h. unter der effektiven Adresse ist nicht der Operand selbst, sondern der höherwertige Teil der 16-bit-Adresse des Operanden abgelegt, unter der folgenden Adresse der niederwertige Teil (*Big-Endian*-Format).

Der aktuelle Inhalt der Register und der Wert des Offsets seien:

BR = $DF80, IR = $80, Offset = $15.

Welche effektive Adresse wird durch einen Lesebefehl mit dieser Adressierungsart angesprochen und welches Byte B wird dadurch aus dem Speicher geholt?

„Zwischenadresse“ = ... = $.....

EA = $....., B = $.....

Hinweis: Der Inhalt der Register BR bzw. IR werde – wie üblich – in den Berechnung für EA mit (BR) und (IR) bezeichnet.

(Lösung auf Seite 170)

Aufgabe 23: Register (einfache Operationen) (I.2.5)

a) Ein 16-bit-Indexregister IR verfüge über die Möglichkeiten der automatischen Modifikation durch Inkrementierung bzw. Dekrementierung um einen wählbaren Offset n sowie der Skalierung mit einem Faktor m. Das Register besitze den momentanen (Hexadezimal-)Wert $0A80. Vom Befehl werde eine skalierte Adressierung mit Postdekrement mit dem Faktor m = 4 und dem Offset n = – 4 ausgeführt. Geben Sie an, welche Speicheradresse durch den Befehl ausgegeben wird und welcher Wert danach im Indexregister steht.

Angesprochene Speicheradresse: EA = ...

Indexregister nach Befehlsausführung: IR = ...

b) Die 8-bit-Register R0, R1, R2 eines Mikroprozessors werden mit den folgenden Werten initialisiert:

R0 = $E5, R1 = $00, R2 = $00.

i. Der Prozessor führe 8-mal hintereinander das folgende Operationspaar aus:

R0 := << R0 *Logisches Linksschieben um 1 bit*

R1 := R1 + R2 + CF *Addition mit Carry Flag*

Welche Werte stehen danach in R0 und R1?

R0 = ... R1 = ...

Welche Zahl gibt der Wert in R1 allgemein an?

ii. Zum Abschluß der genannten Operationsfolge werde nun zusätzlich die folgende Operation ausgeführt wird:

R1 := R1 ^ #$01 *Und-Verknüpfung mit der Konstanten $01.*

Welcher Wert steht danach in R1?

Um welchen Wert in R1 handelt es sich allgemein?

iii. Was ändert sich, wenn man in i) den Befehl zum logischen Linksschieben durch den Befehl zur Rotation nach links mit *Carry Flag* ersetzt?

iv. Was ändert sich, wenn man in i) den Befehl zum logischen Linksschieben durch den Befehl zur Rotation nach links durchs *Carry Flag* ersetzt?

(Lösung auf Seite 170)

Aufgabe 24: Systembus-Multiplexer (I.2.6)

Bild I.2.6-16 im Unterabschnitt I.2.6.5 zeigt die Schnittstelle eines Multiplexbusses.

a) Skizzieren Sie den Aufbau des in diesem Bild beschriebenen Adreßbuspuffers mit Multiplexer für je ein Bit des Programmzählers bzw. des Adreßpuffers.

b) Skizzieren Sie für den betrachteten Multiplexbus den Aufbau des Multiplexers für je ein Bit des Datenbuspuffers sowie des Adreßbuspuffers. Die dazu benötigten bidirektionalen Datenbustreiber stehen als Bausteine zur Verfügung.

(Lösung auf Seite 171)

Aufgabe 25: Busarbiter (I.2.6)

Gegeben sei ein Parallelbus-System mit vier angeschlossenen Knoten K0, ..., K3, die als *Bus Master* auf den Bus zugreifen können. Die Buszuteilung werde durch einen zentralen Arbiter nach dem Verfahren der unabhängigen Anforderung *(Independent Request* vorgenommen. Dazu verfügt jeder Knoten i über zwei Steuerleitungen: BRQi *(Bus Request)* und BGRi *(Bus Grant)*. Der Arbiter realisiere ein Zuteilungsverfahren nach folgenden Vorgaben:

- Jeder Knoten kann jederzeit über seine BRQ-Leitung (BRQ = 1) den Buszugriff anfordern und erhält ggf. über die BGR-Leitung (BGR = 1) den Zugriff zugeteilt. Während des gesamten Zugriffs hält der Knoten sein BRQ-Signal aktiv.
- Bei gleichzeitigen Zugriffen geben die Nummern der Knoten in absteigender Reihenfolge die Prioritäten des Buszugriffs an, d.h. Knoten 0 hat die höchste Priorität, Knoten 3 die niedrigste.
- Ein Knoten bekommt frühestens dann das Zugriffsrecht eingeräumt, wenn der zuletzt zugriffsberechtigte Knoten den Bus durch Rücknahme seines BRQ-Signals freigibt.
- Solange keine neue Anforderung vorliegt, bleibt der Bus bei dem zuletzt zugriffsberechtigten Knoten „geparkt", d.h. sein BGR-Signal bleibt aktiviert und er kann sofort wieder auf den Bus zugreifen – parallel zur Ausgabe seines BRQ-Signals.

a) Gesucht ist ein synchrones Schaltwerk aus logischen Grundschaltungen (Inverter, AND, NAND, OR, NOR, XOR bzw. Antivalenz, Äquivalenz) und getakteten D-Flipflops. Realisieren Sie die Schaltung aus „übersichtlichen" Teilmodulen, indem Sie:

   - zunächst den Prioritätendecoder entwickeln,
   - ein getaktetes Register aus vier D-FFs zur Ausgabe der BGR-Signale aufbauen,
   - eine Steuerlogik entwerfen, die den Takt genau dann zum Register durchschaltet, wenn wenigstens ein Knoten j den Buszugriff verlangt (BRQj = 1) und
     - ein Knoten i das Zugriffsrecht hat (BGRi = 1), dieses aber nicht mehr wieter anfordert (BRQi = 0), oder
     - nach dem Einschalten der Spannungsversorgung noch kein Knoten den Zugriff hatte.

   Geben Sie einen Schaltplan des gesamten Schaltwerks an und beschreiben Sie die Funktion der Schaltung und ihrer Module.

b) Ergänzen Sie Ihre Arbiter-Schaltung nun um eine Logik zur Ausgabe eines Signals BGACK (bzw. BUSY), über das jedem Konten angezeigt wird, daß der Bus einem momentan den Zugriff anfordernden Knoten zugeteilt ist.

Hinweis: Es reicht, eine „intuitive" Lösung anzugeben. Eine Lösung mit Methoden des formalen Schaltwerksentwurfs – Zustandsdiagramm, Übergangstabelle usw. – ist hier nicht zu empfehlen.

(Lösung auf Seite 173)

Aufgabe 26: Prioritätendecoder (I.2.6)

Bild 5 stellt die Methode der unabhängigen, parallelen Busanforderung *(Independent Request)* dar.

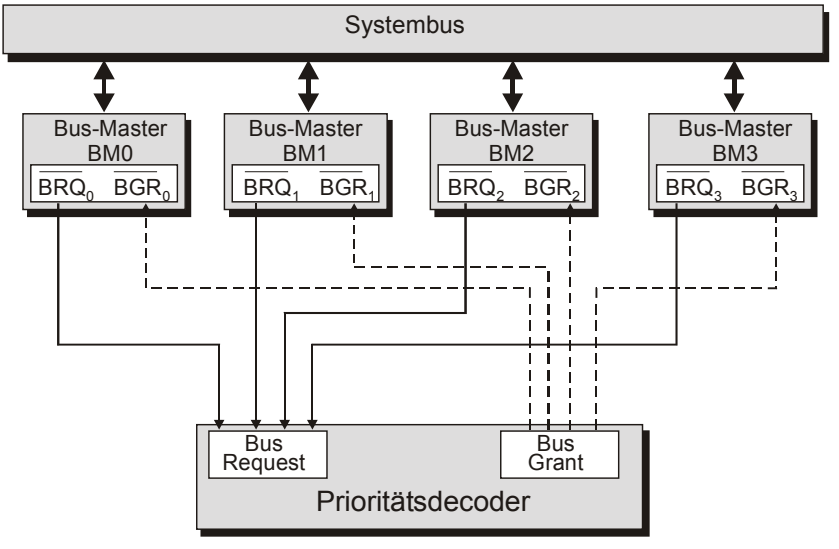


Bild 5: Prinzip der unabhängigen Anforderung

Der Zugriff der vier *Bus Master* BM0, BM1, BM2, BM3 auf den gemeinsamen Systembus wird von einem Prioritätsdecoder gesteuert. Die Priorität sinkt mit steigender Nummer des *Bus Masters* (BM0: höchste Priorität, BM3: niedrigste Priorität).

a) Entwerfen Sie aus logischen Grundschaltungen einen Prioritätendecoder, der das Zugriffsrecht je nach Priorität vergibt. Geben Sie die Funktionsgleichungen für $BGR_i\#$ an und zeichnen Sie ein Blockschaltbild.

b) Ergänzen Sie Ihre Schaltung aus a) zu einem synchronem Schaltwerk, das mit jedem Tatzyklus eine Vergabe des Buszugriffs vornimmt. Eine Neuvergabe findet nur dann statt, wenn:
   - der Bus augenblicklich nicht belegt ist oder
   - der Bus zwar belegt ist, jedoch der zur Zeit zugreifende Master für den nächsten Taktzyklus keinen Zugriff mehr beantragt; d.h., das BRQ-Signal dieses Masters ist inaktiv.

(Lösung auf Seite 174)

Aufgabe 27: Systembus (Übertragungen) (I.2.6)

a) Ein Prozessor führt die Transaktionen Ti auf dem Systembus in verschiedenen Phasen aus, wie es im Bild 6 für Leseoperationen im Burst-Modus dargestellt ist. Jeder Burst besteht aus vier Datentransfers über einen 64-bit-Datenbus.

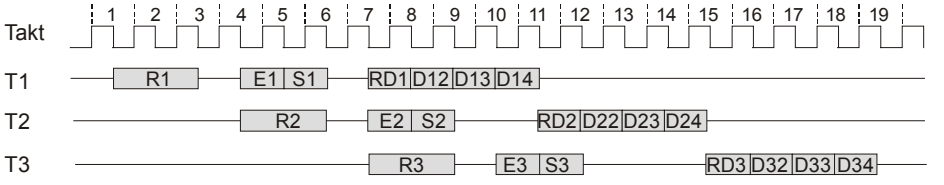


Bild 6: Phasen-Zeitdiagramm der Burst-Übertragung

Im Bild stehen

Ri für die *Request*-Phasen,
Ei für die *Error*-Phasen,
Si für die *Snoop*-Phasen,
RDi für die simultanen *Response*-/Datenphasen,
Dij für die anderen Datenphasen der Transaktionen Ti (i = 1, 2, 3, j = 2, 3, 4).

- In der *Request*-Phase wird die Art der gewünschten Transaktion (Schreiben, Lesen, *Interrupt Acknowledge* usw.) festgelegt und die Adresse ausgegeben.
- In der *Error*-Phase wird ein eventuell während der *Request*-Phase aufgetretener Paritätsfehler gemeldet.
- In der *Snoop*-Phase wird angezeigt, ob das angesprochene Datum bereits im Cache abgelegt ist oder nicht.
- In der *Response*-Phase wird angezeigt, ob die Transaktion erfolgreich war oder mit einem Fehler abgeschlossen wurde.
- In den Datenphasen werden die Daten übertragen. Dabei wird bei Lesezugriffen die erste Datenphase überlappend mit der *Response*-Phase ausgeführt.

Zeichnen Sie den Ablauf der Transaktionen in das „übliche“ Zeitdiagramm der Bussignale um, das im Bild 7 vorgegeben ist. Darin sind die für die verschiedenen Phasen der Transaktionen benötigten Signale zu Signalgruppen zusammengefaßt. Außerdem wird unterstellt, daß keine Fehler auftreten, die *Error*-Phase also nicht ausgewertet werden muß.

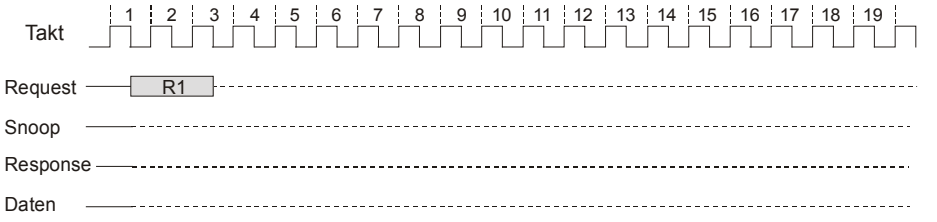


Bild 7: Zeitdiagramm der Bussignale für die Burst-Übertragung

Geben Sie an, wie viele Taktzyklen die Ausführung der Transaktionen Ti jeweils dauert, wenn man die Zyklen, in denen die *Request*-Phasen beginnen und in denen die Datenphasen enden, mitzählt.

T1: ....., T2: ....., T3: .....

b) Vervollständigen Sie nach dem in a) dargestellten Muster das im Bild 8 vorgegebene Zeitdiagramm für eine ununterbrochene Folge von Burst-Übertragungen. Die Anzahl der ausstehenden Transaktionen sei dazu als unbegrenzt angenommen.

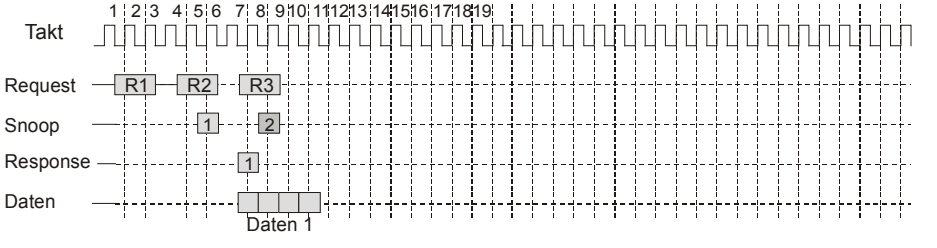


Bild 8: Zeitdiagramm für ununterbrochene Burst-Übertragungen

Leiten Sie aus dem Diagramm (heuristisch) eine Formel für die Dauer der Transaktion Tn ab und diskutieren Sie das damit in Verbindung stehende Problem für die Verwaltung der Transaktionen im Prozessor.

Zeit für Tn: ..... (für n = 1, ..., 6)

Formel: ..... (für n allgemein)

Problem: .....

c) Nun sei vorausgesetzt, daß der Prozessor nur maximal vier ausstehende Transaktionen verwalten kann, d.h. max. vier Transaktionen können gleichzeitig in Bearbeitung und noch nicht beendet sein. Dazu besitzt der Prozessor einen als Warteschlange *(In-Order-Queue,* FIFO – *First-in, First-out)* organisierten Puffer, in

dem die aktiven Transaktionen eingetragen werden. Ergänzen Sie das vorgegebene Zeitdiagramm im Bild 9 und tragen sie die jeweils in der FIFO gespeicherten Transaktionen ein.

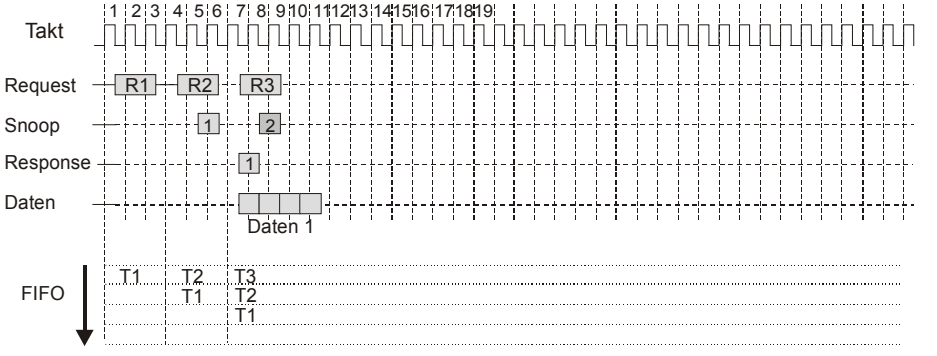


Bild 9: Zeitdiagramm für ununterbrochene Burst-Übertragungen

Es werde dabei vorausgesetzt, daß

- der Eintrag einer Transaktion in die FIFO in dem Taktzyklus geschieht, in dem ihre *Request*-Phase beginnt,
- das Entfernen einer Transaktion aus der FIFO mit dem letzten Taktzyklus der Datenphase vorgenommen wird,
- eine neue Transaktion erst dann begonnen werden kann, wenn in der FIFO (wenigstens) eine Position frei ist.

Geben Sie für die Transaktionen T1 – T8 die Ausführungsdauer in Taktzyklen an und leiten Sie daraus (heuristisch) eine allgemeine Formel für die Ausführungsdauer der Transaktion Tn her.

T1: ....., T2: ....., T3: ....., T4: .....,

T5: ....., T6: ....., T7: ....., T8: .....

Allgemeine Formel für die Ausführungsdauer von Tn:

.....

d) Geben Sie für einen 66-MHz-Takt des Systembusses die erreichbaren Übertragungsraten in MByte/s an, wenn

- nur auf einzelne 64-bit-Speicherwörter in nicht überlappenden Transaktionen,
- nur im unter c) dargestellten ununterbrochenen Burst-Modus

zugegriffen wird.

(Lösung auf Seite 176)

Aufgabe 28: Systembus (Zeitverhalten) (I.2.6)

Ein Mikroprozessor kommuniziere mit einer Peripheriekomponente über einen asynchronen Systembus. Zur Synchronisation der Kommunikation im Handshake-Verfahren besitze der Prozessor das Signal REQ *(Request)*, die angesprochene Komponente das Signal ACK *(Acknowledge)*, die beide im ‚0'-Pegel aktiv sind. Tabelle 3 gibt die Funktion dieser Signale wieder.

Tabelle 3: Funktion der Signale REQ und ACK

Signal	negative Flanke	positive Flanke	
		Lesen	Schreiben
REQ	Komponenten-Selektion	Datum empfangen	Datenübernahme erkannt
ACK	Selektion erkannt	Datum bereitgestellt	Datum übernommen

Im Unterschied zur vereinfachenden Darstellung der asynchronen Übertragung im Buchtext sollen in dieser Aufgabe nun die folgenden realistischen Parameter berücksichtigt werden, die in den Zeitdiagrammen eingezeichnet sind:

- **Einschwingzeiten:** Signalwechsel auf den Daten- und Adreßleitungen benötigen eine bestimmte Zeit, bis ein stabiler Zustand eingenommen ist. Das gelte auch für einen Wechsel in den Tristate-Zustand.
- **Signallaufzeiten:** Alle Signale benötigen eine gewisse Zeit, bis sie auf dem Systembus von der Quelle zum Ziel gelaufen sind. In diesen Zeiten sei die Reaktionszeit des Signalempfängers mit berücksichtigt.
- **Arbeitstakt des Mikroprozessors:** Die Zustandswechsel der Ausgangssignale des Prozessors werden durch die positive Flanke des Takts ausgelöst und müssen z.T. mit einer Verzögerung auftreten, die die o.g. Einschwingzeiten berücksichtigt. Die Übernahme der Daten in den Prozessor wird ebenfalls durch die positiven Taktflanken veranlaßt.
- **Zugriffszeit:** Die Peripheriekomponente kann frühestens nach dieser Zeit ein angebotenes Datum übernehmen bzw. mit der Ausgabe eines angeforderten Datums beginnen. Sie stimme hier – vereinfachend – mit der Taktzykluszeit des Prozessors überein.

Benutzen Sie für die geforderten Lösungen die im Bild 10 angegebenen Symbole bzw. Zeitangaben.

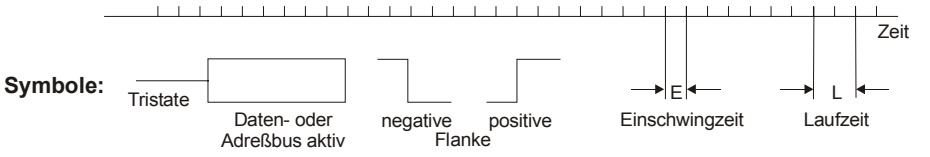
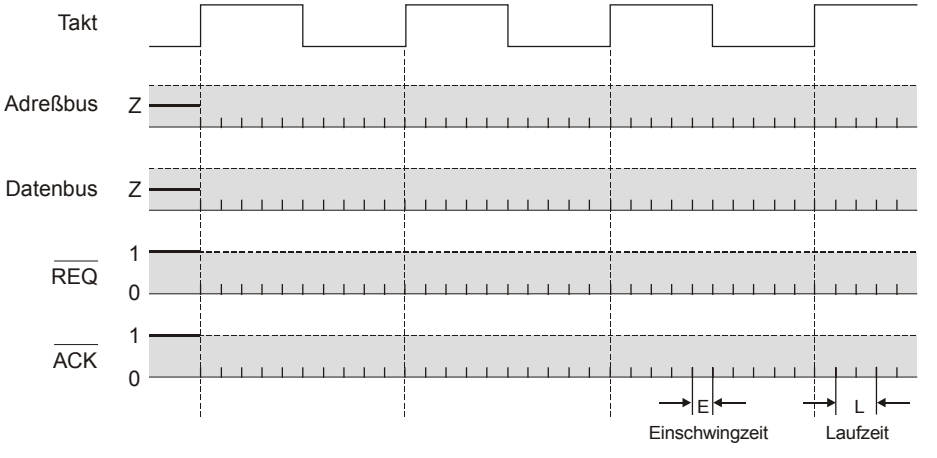


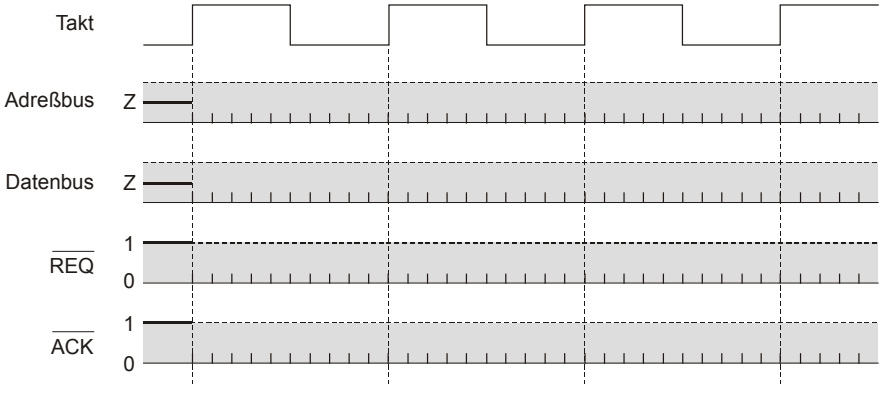
Bild 10: Symbole zur Darstellung der Zeitbedingungen

a) Tragen Sie in das folgende Diagramm den zeitlichen Verlauf der Adreß-, Daten- und Handshake-Signale für das **Lesen eines Datums** von der Peripheriekomponente ein. Geben Sie dabei für die Signalwechsel die frühest möglichen Zeitpunkte an und berücksichtigen Sie dabei die eingezeichneten Parameter. Kennzeichnen Sie die zugrunde liegenden Kausalbeziehungen durch Pfeile und numerieren Sie sie. Geben Sie für jede Kausalbeziehung und ihre Auswirkung eine kurze Interpretation an.



1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....

b) Tragen Sie nun in das folgende Diagramm den zeitlichen Verlauf der genannten Signale für das **Schreiben eines Datums** in die Peripheriekomponente ein. Gehen Sie dazu völlig analog zum Punkt a) vor.



1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. .....

c) Eine „langsame“ Peripheriekomponente benötige sehr viel Zeit, bevor sie ein gefordertes Datum liefern bzw. annehmen kann. Ist bei der dargestellten Form der Handshake-Synchronisation durch die Signale REQ und ACK ein zusätzliches Signal erforderlich, mit dem die Komponente vom Prozessor das Einfügen von Wartezeiten anfordern kann? Begründen Sie Ihre Antwort.

(Lösung auf Seite 177)

I.3 Hardware/Software-Schnittstelle

Aufgabe 29: Integer-Zahlen (Zahlenbereich) (I.3.1)

Bestimmen Sie näherungsweise für 32- und 64-bit-Integer-Zahlen mit bzw. ohne Vorzeichen den darstellbaren Zahlenbereich.

(Lösung auf Seite 179)

Aufgabe 30: Integer-Zahlen (Vorzeichenerweiterung) (I.3.1)

Die zwei vorzeichenbehafteten 8-bit-Zahlen A_8 = \$63, B_8 = \$A7 (im Zweierkomplement) sollen so in 16-bit-Zahlen A_{16}, B_{16} umgewandelt werden, daß sich ihr Wert nicht ändert. Geben Sie A_{16}, B_{16} an und begründen Sie Ihre Lösung!

(Lösung auf Seite 179)

Aufgabe 31: Gleitpunktzahlen (8 bit) (I.3.1)

Gegeben sei das im Bild 11 dargestellte Format einer 8-bit-Gleitpunktzahl.

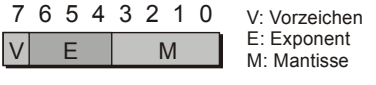


Bild 11: Ein 8-bit-Gleitpunktformat

Der Dezimalwert einer Zahl Z ergibt sich aus der Formel

$$Z = (-1)^V \cdot 2^{E-3} \cdot (1.M)_2$$

für alle möglichen binären Belegungen der Felder V, E, M.

a) Berechnen Sie den Dezimalwert der Belegung 1010 0100.

b) Wie lautet die größte Dezimalzahl, die mit diesem Format darstellbar ist?

c) Wie lautet die kleinste positive Dezimalzahl, die mit diesem Format darstellbar ist?

d) Welche „wichtige“ Dezimalzahl kann mit diesem Format nicht dargestellt werden?

(Lösung auf Seite 179)

Aufgabe 32: IEEE-754-Format (Zahlenbereich) (I.3.1)

a) Bestimmen Sie für die 32– bzw. 64-bit-Gleitpunkt-Zahlenformate des IEEE-754-Standards den darstellbaren (absoluten) Zahlenbereich.

b) Stellen sie die Zahlen $Z1_{10} = -1{,}75$ und $Z2_{10} = 36.864$ im 32-bit-Zahlenformat als Binärzahl dar.

(Lösung auf Seite 180)

Aufgabe 33: IEEE-754-Format (Addition) (I.3.1)

Die folgenden zwei Dezimalzahlen sollen im 32-bit-Format des IEEE-754-Standards dargestellt werden:

$$Z1_{10} = 5{,}875,\ Z2_{10} = -3{,}125.$$

a) Geben Sie Charakteristik C, Exponent E und Mantisse M in hexadezimaler, binärer und dezimaler Form an.

Z1: Vorzeichen: VZ = ...

Charakteristik: C = $..... = ...$_{10}$

Exponent: E = $..... = ...$_{10}$

Mantisse: $(1.M)_2$ = (.....)$_2$ = ...$_{10}$

Z2: Vorzeichen: VZ = ...

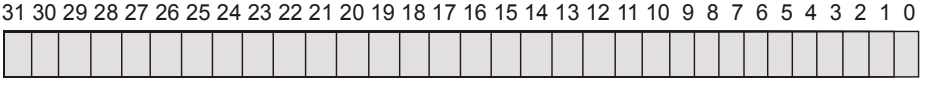
Charakteristik: C = $..... = ...$_{10}$

Exponent: E = $..... = ...$_{10}$

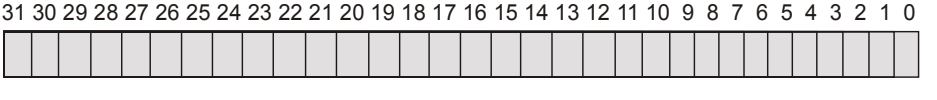
Mantisse: $(1.M)_2$ = (.....)$_2$ = ...$_{10}$

b) Tragen Sie diese Zahlen in binärer und hexadezimaler Form ein. Kennzeichnen Sie die unterscheidbaren Bitfelder sowie die Lage des Punktes (Kommas).

$Z1_{32}$ = $.....

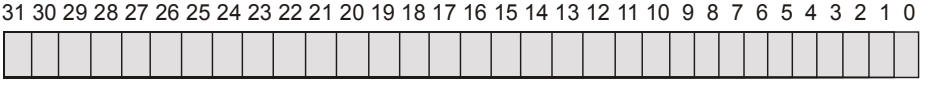


$Z2_{32}$ = $.....



c) Es soll die Summe S der Zahlen $Z1_{32}$ und $Z2_{32}$ im IEEE-54-Format gebildet werden. Wie erfolgt die Angleichung der Charakteristik von $Z2_{32}$? Geben Sie die Mantisse von $Z2_{32}$ vor der Ausführung der Addition als hexadezimale Zahl an. Führen Sie die Addition durch und tragen Sie die Summe ein.

Mantisse von $Z2_{32}$ vor der Addition: \$.....



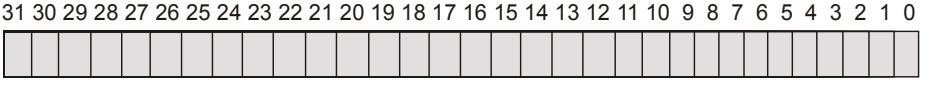
$Z1_{32} + Z2_{32}$ = ...

(Lösung auf Seite 180)

Aufgabe 34: IEEE-754-Format (IEEE → dezimal) (I.3.1)

Z_{32} sei eine 32-bit-Zahl nach dem IEEE-754-Standard, deren hexadezimale Darstellung gegeben sei durch: Z_{32} = \$9F5A 0000

a) Tragen Sie Z_{32} in binärer Form in den folgenden Bitrahmen ein. Kennzeichnen und benennen Sie die unterscheidbaren Bitfelder.



.....

b) Geben Sie die Zahl Z_{32} als Dezimalzahl Z_{10} an (VZ: Vorzeichen, C: Charakteristik, E: Exponent, $..._{32}$: Zahl im IEEE-754-Format, $..._{10}$: Dezimalzahl, $..._2$: Binärzahl). Dabei soll die Mantisse (mit ihren signifikanten Stellen) zunächst in binärer, dann in dezimaler Form angegeben werden. Das Endergebnis soll in Exponentschreibweise zur Basis 10 dargestellt werden (3 Nachpunktstellen).

$Z_{32} = (-1)^{VZ} \cdot 2^{C} \cdot (1.M)_2 = (-1)^{\cdots\cdots} \cdot 2^{\cdots\cdots\cdots} \cdot (\ldots\ldots\ldots\ldots\ldots\ldots)_2 \quad \Rightarrow$

$Z_{10} = (-1)^{VZ} \cdot 2^{E} \cdot (1.M)_{10} = (-1)^{\cdots\cdots} \cdot 2^{\cdots\cdots\cdots} \cdot \ldots\ldots\ldots\ldots\ldots\ldots_{10}$

$\approx$ ..... (zur Basis 10)

c) Durch die beiden Befehle FGETEXP, FGETMAN werden für eine Gleitpunktzahl der Exponent $\mathscr{E}$ (nicht die Charakteristik) bzw. die Mantisse $\mathscr{M}$ (einschließlich ihres Vorzeichens) nach dem IEEE-754-Standard selbst als 32-bit-Zahlen nach diesem Standard dargestellt. Wenden Sie diese beiden Befehle auf die oben angegebene Zahl Z_{32} an. (Verwenden Sie zur Hilfe die Bitdarstellungen.)

FGETEXP

$\mathscr{E}_{10}$ = ...$_{10}$ = $(-1)^{VZ} \cdot 2^{E} \cdot (1.M)_{10}$ = $(-1)^{\cdots} \cdot 2^{\cdots} \cdot$ .....$_{10}$ $\Rightarrow$

$\mathscr{E}_{32} = (-1)^{VZ} \cdot 2^{C} \cdot (1.M)_2 = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2$

= \$.....

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

FGETMAN

$\circlearrowright\mathscr{M}_{10}$ = ...$_{10}$ = $(-1)^{VZ} \cdot 2^{E} \cdot (1.M)_{10}$ = $(-1)^{\cdots} \cdot 2^{\cdots} \cdot$ .....$_{10}$ $\Rightarrow$

$\circlearrowright\mathscr{M}_{32} = (-1)^{VZ} \cdot 2^{C} \cdot (1.M)_2 = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2$

= \$.....

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

(Lösung auf Seite 181)

Aufgabe 35: IEEE-754-Format (Gleitpunkt-Dezimalzahl) (I.3.1)

a) Stellen Sie die Dezimalzahl $Z_{10} = -22{,}5$ im 32-bit-Format des IEEE-754-Standards in binärer Form dar. Kennzeichnen Sie die unterscheidbaren Bitfelder (Vorzeichen, Charakteristik, Mantisse). Wie sieht die hexadezimale Darstellung aus?

$Z_{32} = (-1)^{VZ} \cdot 2^{C} \cdot (1.M)_2$ = ..., Z_{32} = \$.....

b) Welche Dezimalzahl Z_{10} ist durch die 32-bit-Gleitpunktzahl Z_{32} = \$3EEC 0000 (nach dem IEEE-754-Standard) gegeben? Geben Sie Z auch als Dezimalzahl in Exponential-Schreibweise zur Basis 2 an.

$Z_{10} = (-1)^{VZ} \cdot 2^{E} \cdot (1.M)_{10} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot 1.\ldots\ldots = \ldots\ldots,\ldots\ldots$

c) Eine Arithmetikeinheit verarbeite gepackte Gleitpunkt-Dezimalzahlen, wie sie im Abschnitt I.3.1 beschrieben wurden. Die Belegung ‚00' im Typfeld kennzeichne dabei jede gültige Zahl des darstellbaren Zahlenbereichs. Der Exponent bezieht sich auf die Basis 10.

i. Geben Sie den positiven bzw. negativen darstellbaren Zahlenbereich durch die jeweils größte und kleinste Zahl (außer der Zahl 0) an.

ii. Welche Dezimalzahl Z_{10} wird durch die folgende Bitfolge dargestellt:

0100000000010110000000000000001100010100000010101
10010010011001010011010110001001011110010011010 1

Stellen Sie zunächst diese Zahl Z als 24-stellige-Hexadezimalzahl, dann als Dezimalzahl dar.

Z = $.....

Z_{10} = ...

Um welche wichtige Zahl handelt es sich?

(Lösung auf Seite 182)

Aufgabe 36: IEEE-754-Format (Konstanten-ROM) (I.3.1)

a) Ein Gleitpunkt-Rechenwerk verarbeite nur das 32-bit-Gleitpunktformat des IEEE-754-Standards. In seinem Konstanten-Festwertspeicher finde sich die folgende Zahl Z_{32}:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	1	0	1	1	0	1	0

Geben Sie zunächst die Zahl Z in hexadezimaler Form an.

Z_{32} = $.....

Kennzeichnen Sie nun die verschiedenen Bitfelder in der oben dargestellten Zahl.

Berechnen Sie (näherungsweise) die Dezimalzahlen-Darstellung dieser Konstanten. Dabei reicht es, die höchstwertigen 7 – 8 Stellen der Mantisse auszuwerten.

$Z_{32} = (-1)^{VZ} \cdot 2^{C} \cdot (1.M)_2 \approx$ ..... $\Rightarrow$

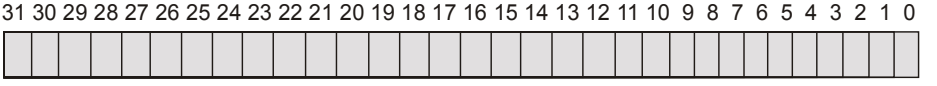
$Z_{10} \approx$ ..... , ...

Um welche wichtige Konstante handelt es sich? .....

b) Wandeln Sie die dezimale Konstante 10 in das 32-bit-Format des IEEE-754-Standards und tragen Sie sie in den folgenden Bitrahmen ein. Kennzeichnen Sie die verschiedenen Bitfelder. Stellen Sie die Gleitpunktzahl als hexadezimale Zahl dar.

$Z_{10} = 10$ = ...$_2$ $\Rightarrow$

$Z_{32} = (-1)^{VZ} \cdot 2^{C} \cdot (1.M)_2$ = ...



Z_{32} = $.....

(Lösung auf Seite 183)

Aufgabe 37: IEEE-754-Format (dezimal → IEEE) (I.3.1)

a) Stellen Sie die Dezimalzahl Z_{10} = +0,453125 im 32-bit-Gleitpunktformat des IEEE-754-Standards als Z_{32} in hexadezimaler Form dar. Tragen Sie Z_{32} dazu in den folgenden Zahlenrahmen ein und kennzeichnen Sie die einzelnen Bitfelder graphisch und durch ihre Bezeichnungen!

Geben Sie Z_{10} dazu zunächst als gebrochene Dezimalzahl in Potenzschreibweise zur Basis 2 an:

$Z_{10} = (-1)^{VZ} \cdot 2^{VZ \cdot E} \cdot (1.M)_{10} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot \ldots\ldots_{10}$

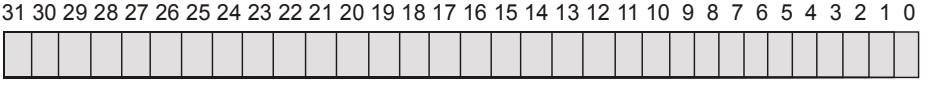
$\Rightarrow \quad Z_{32} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2$

Z_{32} = $ .....

b) Stellen Sie die Zahl Z_{10} = –464 im 32-bit-Gleitpunktformat des IEEE-754-Standards in hexadezimaler Form Z_{32} dar (Bitte Rechnung angeben!). Tragen Sie Z_{32} in den folgenden Zahlenrahmen ein und kennzeichnen Sie die einzelnen Bitfelder!

$Z = (-1)^{VZ} \cdot 2^{VZ \cdot E} \cdot 1.M_{10} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot \ldots\ldots_{10}$

$\Rightarrow \quad Z_{32} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2$



Z_{32} = $.....

c) Stellen Sie die Zahl Z = –464 im 64-bit-Gleitpunktformat des IEEE-754-Standards in hexadezimaler Form dar. (Bitte Rechnung angeben!)

$Z_{64} = (-1)^{\cdots} \cdot 2^{\cdots} \cdot (\ldots\ldots)_2$

Z_{64} = $.....

(Lösung auf Seite 184)

Aufgabe 38: IEEE-754-Format (Gleitpunkt-Multiplikation) (I.3.1)

Für das Produkt zweier Gleitpunktzahlen $Z1 = M1 \cdot 2^{E1}$, $Z2 = M2 \cdot 2^{E2}$ gilt bekanntlich:

$$Z = Z1 \cdot Z2 = M1 \cdot M2 \cdot 2^{E1+E2}$$

(Darin seien mit M1, M2 die vorzeichenbehafteten Mantissen und E1, E2 die vorzeichenbehafteten Exponenten benannt.)

a) Stellen Sie zunächst die beiden folgenden Zahlen im 32-bit-Format des IEEE-754-Standards in „Potenzschreibweise“ dar:

$Z1_{32}$ = \$C2D8 0000 = $(-1)^{\dots\dots} \cdot (\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots)_2 \cdot 2^{\dots\dots\dots}$

$Z2_{32}$ = \$39B4 0000 = $(-1)^{\dots\dots} \cdot (\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots)_2 \cdot 2^{\dots\dots\dots}$

b) Berechnen Sie nun „schriftlich“ das Produkt der Mantissen:

$(1.M1)_2 \cdot (1.M2)_2 = (\dots\dots\dots\dots\dots\dots\dots\dots)_2 = (1.\dots\dots\dots\dots\dots)_2 \cdot 2^{\dots\dots\dots}$

c) Ermitteln Sie jetzt die Charakteristik des Produkts und berücksichtigen Sie dabei das Ergebnis der Multiplikation der Mantissen!

C = ...

d) Geben Sie nun das Produkt Z in „Potenzschreibweise“ sowie in binärer und hexadezimaler Form an.

$Z_{32} = (-1)^{\dots\dots} \cdot (\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots)_2 \cdot 2^{\dots\dots\dots}$

$Z_{32} = (\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots)_2$

Z_{32} = \$...

(Lösung auf Seite 184)

Aufgabe 39: IEEE-754-Format (Umwandlung: 32 bit → 64 bit) (I.3.1)

a) Geben Sie allgemein an, wie eine Gleitpunktzahl nach dem 32-bit-IEEE-Format in eine 64-bit-IEEE-Gleitpunktzahl umgewandelt werden kann.

b) Demonstrieren Sie das Vorgehen am Beispiel der Zahl Z_{32} = \$C3B4 0000.

(Lösung auf Seite 185)

Aufgabe 40: Festpunktzahlen (einfache Beispiele) (I.3.1)

Ein 16-bit-Festpunktrechenwerk verarbeite vorzeichenbehaftete gebrochene Zahlen im 1.15-Format (1 Vorkomma-, 15 Nachkommastellen) nach der Skizze in Bild 12.

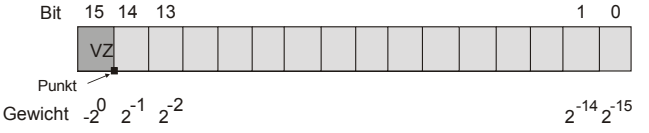


Bild 12: 1.15-Format

In der Skizze ist für jedes Bit seine Wertigkeit angegeben.

a) Geben Sie für das beschriebene Format den darstellbaren Zahlenbereich an.

b) Welche Dezimalzahlen werden durch folgende, in hexadezimaler Darstellung gegebenen Festpunktzahlen (Format 1.15) repräsentiert:

Z1 = $7000 Z2 = $5B00

Z3 = $9000 Z4 = $FFFF

c) Stellen Sie die folgenden Zahlen als 16-bit-Festpunktzahlen in hexadezimaler Form dar:

Z1 = – 0,5625 Z2 = – 1,0

Z3 = 0,5625 Z4 = 0,0625

(Lösung auf Seite 185)

Aufgabe 41: Festpunktzahlen (Komplement und Runden) (I.3.1)

a) Z sei eine 16-bit-Festpunktzahl im 1.15-Format (nach Bild 12). Man kann zeigen, daß man die zu Z komplementäre Zahl –Z nach folgendem Verfahren gewinnen kann:

- Invertiere jedes Bit von Z.
- Addiere zum Ergebnis der Inversion den Wert des LSB *(Least Significant Bit)*.[1]

Überprüfen Sie dieses Verfahren an den beiden in dezimaler Form gegebenen Zahlen $Z_1 = 0{,}3125_{10}$ und $Z_2 = -0{,}25_{10}$.

Z_1: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15)

Z_{1inv}: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15)

\+ LSB: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15)

$-Z_1$: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15) = ... , ...$_{10}$

[1] Dieses Verfahren entspricht der Umwandlung einer negativen Integer-Zahl in ihr Zweierkomplement.

Z_2: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15)

Z_{2inv}: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15)

\+ LSB: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15)

$-Z_2$: _ . _ _ _ _ _ _ _ _ _ _ _ _ _ _ (1.15) = ... , ...$\cdot 10$

b) Leiten Sie das Verfahren nach a) mathematisch her. Beachten Sie dabei, daß für einen Binärwert $d \in \{0, 1\}$ gilt: $1 - d \mathrel{\hat{=}} \overline{d}$. Gehen Sie von folgender trivialen Aussage aus, indem Sie die aufgeführten Zahlen als Summen von Potenzen von 2 mit den entsprechenden Koeffizienten darstellen: $-Z = 1 - 1 - Z = 1 + (-1) + (-Z)$. Benutzen Sie folgende binäre Darstellung der Zahl Z: $Z = (d_0 . d_1 d_2 ... d_{15})_2$.

1 = ...

–1 = ...

–Z = ...

–Z = 1 + (–1) + (–Z) = ...

.....

.....

c) Ein Festpunkt-Rechenwerk arbeite intern mit einer erhöhten Genauigkeit, indem es eine im 1.15-Format gegebene Zahl Z um einen Teil R aus vier Bits nach folgendem Bild 13 zum 1.19-Format mit den angegebenen Wertigkeiten ergänzt:

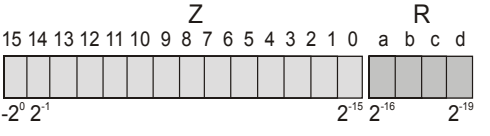


Bild 13: Das erweiterte 1.19-Format

Jedes berechnete Ergebnis E wird nach folgender Vorschrift ins 1.15-Format gerundet:

- $E \rightarrow Z$ falls $R < 2^{-16}$ (nächst kleinere 1.15-Zahl)
- $E \rightarrow Z + 1 \cdot 2^{-15}$ falls $R \geq 2^{-16}$ (nächst größere 1.15-Zahl).

i. Geben Sie für das betrachtete Rundungsverfahren die möglichen Belegungen des Restes R ($\neq 0000_2$) an, für die ein Ergebnis aufwärts bzw. abwärts gerundet wird:

aufwärts: R = ..., ..., ..., ..., ..., ..., ..., ...

abwärts: R = ..., ..., ..., ..., ..., ..., ..., ...

ii. Führen Sie diese Rundung an den folgenden Beispielen durch:

$E_1 = 1.011100111010111\ 1000_{(1.19)} \rightarrow Z_1 = \_\ .\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \__{(1.15)}$

$E_2 = 0.101101100111001\ 0101_{(1.19)} \rightarrow Z_2 = \_\ .\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \__{(1.15)}$

$E_3 = 1.001001110100100\ 1101_{(1.19)} \rightarrow Z_3 = \_\ .\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \__{(1.15)}$

iii. Geben Sie an, durch welche einfache arithmetische Operation man dieses Rundungsverfahren realisieren kann! (Hinweis: Denken Sie an das „kaufmännische Runden", das Sie in der Schule gelernt haben.)

d) Betrachten Sie weiter das in c) beschriebene Rundungsverfahren. In einer Folge von Ergebnissen mit gleichmäßig verteilten Resten R führt diese Verfahren zu einer „positiven" Abweichung *(biased rounding)*, da der mittlere Wert $R = 1000_2$ stets nach oben gerundet wird. Dies wird durch die folgende Modifikation des Rundungsverfahrens vermieden *(unbiased rounding):*

- Verfahre zunächst wie in c).
- Nur für $R = 1000_2$ setze Bit 0 (LSB) im Rundungsergebnis auf den Wert 0.

i. Wenden Sie dieses Rundungsverfahren auf die beiden folgenden Ergebnisse im 1.19-Format an:

$E_1 = 1.100111011011111\ 1000_{(1.19)} \rightarrow Z_1 = \_\ .\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \__{(1.15)}$

$E_2 = 0.001110011011010\ 1000_{(1.19)} \rightarrow Z_2 = \_\ .\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \_\ \__{(1.15)}$

ii. Geben Sie an, ob ein Ergebnis mit $R = 1000_2$ aufwärts oder abwärts gerundet wird, wenn es, aufgefaßt als 16-bit-Integerzahl Z, gerade bzw. ungerade ist.

- gerade: .....
- ungerade: .....

(Lösung auf Seite 186)

Aufgabe 42: Festpunktzahlen (Vergleich) (I.3.1)

a) Für vorzeichenbehaftete Festpunktzahlen Z im 1.15-Format gilt die folgende Darstellung in Potenzschreibweise zur Basis 2:

$$Z = d_0 \cdot (-2^0) + d_1 \cdot 2^{-1} + d_2 \cdot 2^{-2} + ...+ d_{15} \cdot 2^{-15}$$

In Binärschreibweise gilt: $Z = d_0.d_1...d_{15}$ mit $d_i \in \{0, 1\}$

Man kann die Zahlen $Z_{(1.15)}$ aber auch als vorzeichenlose ganze Zahlen *(Integer)* interpretieren und erhält dann:

$$Z_I = d_0d_1...d_{15} \quad \text{mit} \quad 0 \le Z_I \le 65535.$$

Dabei ist $d_0 = MSB(Z_I)$ das höchstwertige Bit. (MSB – *Most Significant Bit*)

Beweisen Sie die folgende Umrechnungsvorschrift für Z aus Z_I:

$$Z = Z_I \cdot 2^{-15} - 2 \cdot d_0$$

Herleitung

.....

.....

.....

.....

.....

b) Der Vergleich zweier Festpunktzahlen Z, Z' kann durch die Vergleicherschaltung (Komparator) einer Integer-ALU durchgeführt werden, indem die beiden Zahlen als vorzeichenlose ganze Zahlen Z_I, Z'_I interpretiert und verglichen werden. Dies sollen Sie hier schrittweise beweisen. Dazu seien $Z \neq Z'$ und wie unter a): $d_0 = MSB(Z_I)$ und $d_0' = MSB(Z'_I)$.

i. Es seien: $d_0 \neq d_0'$, d.h. die beiden Zahlen Z, Z' haben entgegengesetzte Vorzeichen.
Zeigen Sie:

$$Z_I > Z'_I \Leftrightarrow Z < Z'$$

.....

.....

.....

.....

ii. Es gelte nun: $d_0 = d_0'$, d.h. die beiden Zahlen Z, Z' haben dasselbe Vorzeichen. Zeigen Sie nun mit der Formel aus a):

$$Z_I > Z'_I \Leftrightarrow Z > Z'$$

.....

.....

.....

.....

.....

c) Die Aussage von b) gilt auch für die Beträge von 32-bit-Gleitpunktzahlen nach dem IEEE-754-Zahlen, d.h. man kann diese dadurch vergleichen, daß man sie als vorzeichenlose ganze Zahlen interpretiert und durch eine Integer-ALU bearbeiten läßt.

i. Geben Sie an, durch welche Operation man den Betrag einer 32-bit-Zahl nach dem IEEE-Standard ermitteln kann.

.....

.....

ii. Gegeben seien nun zwei 32-bit-Zahlen Z, Z' und ihre Beträge $|Z|$, $|Z'|$. Die Beträge werden wie unter a) als vorzeichenlose ganze Zahlen interpretiert und dann mit $|Z|_I$, $|Z'|_I$ bezeichnet. Zeigen Sie:

$$|Z|_I > |Z'|_I \Leftrightarrow |Z| > |Z'|$$

Das heißt, wenn die als vorzeichenlose ganze Zahlen aufgefaßten Beträge der Zahlen in einem bestimmten Größenverhältnis stehen, so stehen auch die Beträge selbst in diesem Verhältnis.

.....

.....

.....

.....

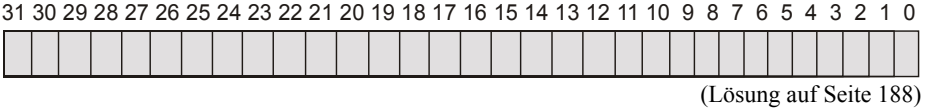
(Lösung auf Seite 187)

Aufgabe 43: Festpunktzahlen (Konvertierung) (I.3.1)

Wandeln Sie die folgende 16-bit-Festpunktzahl Z direkt in eine 32-bit-Gleitpunktzahl Z' nach dem IEEE-754-Format um, ohne den Umweg über die Berechnung des Dezimalwertes zu gehen. Geben Sie den Lösungsweg an.

$Z = \$1750 = (.....)_2 \quad \Rightarrow \quad Z' = \$.....$

Tragen Sie Z' in den Bitrahmen ein und kennzeichnen Sie das Vorzeichen, die Charakteristik und die darzustellende Mantisse.



(Lösung auf Seite 188)

Aufgabe 44: Festpunktzahlen (Konvertierung) (I.3.1)

a) Geben Sie allgemein an, wie eine negative, nach Betrag und Vorzeichen angegebene binäre 16-bit-Festpunktzahl $Z = -0.d_1....d_{15}$ in eine vorzeichenbehaftete Festpunktzahl im 1.15-Format umgerechnet werden kann. (Berücksichtigen Sie dabei das LSB = 2^{-15}.)

b) Demonstrieren Sie das Vorgehen am Beispiel der Festpunktzahl:
$Z = -0.101101000000000$.

(Lösung auf Seite 188)

Aufgabe 45: Dynamik der Zahlenbereiche (I.3.1)

a) Vorzeichenbehaftete Festpunktzahlen der Länge n bit seien im Format (1.n-1) gegeben, d.h. eine Stelle vor dem Dezimalpunkt, n-1 Stellen dahinter.

Geben Sie in allgemeiner Form die kleinste und die größte darstellbare positive (>0) Festpunktzahl in binärer Form ($XX.....XX_2$) und in Potenzen von 2 an:

$Z_{fix,min}$ = ...$_2$ = ... (Potenz von 2)

$Z_{fix,max}$ = ...$_2$ = ... (Potenz von 2)

Welchen Wert haben beide Zahlen für n = 32, also 32-bit-Festpunktzahlen, in Potenzen von 2?

$Z_{fix,min}$ = ... ; $Z_{fix,max}$ = ... ≈ .....

b) Geben Sie für (normalisierte) 32-bit-Gleitpunktzahlen nach dem IEEE-754-Standard den minimalen und maximalen (positiven) Wert für die Mantisse in binärer Form und in Potenzen von 2 an:

$(1.M)_{min}$ = ...$_2$ = ...

$(1.M)_{nax}$ = ...$_2$ = ... ≈ .....

Berechnen Sie daraus (näherungsweise) die kleinste bzw. größte positive (>0) Zahl des IEEE-Standards. Beachten Sie dabei, daß für die 8-bit-Charakteristik C gilt:

$$1 \le C \le 254!$$

Z_{min} = ... · $2^{.....}$ = ...

Z_{max} = ... · $2^{.....}$ = ...

c) Bekanntermaßen gilt für die Umrechnung von Potenzen von 2 in Potenzen von 10 die Beziehung (vgl. Aufgabe 1):

$2^n = 10^x \Rightarrow x = n \cdot \log_{10} 2 \approx 0.301 \cdot n$ ($\log_{10}$: Logarithmus zur Basis 10)

Unter dem **Dynamikbereich** einer Zahlendarstellung versteht man das Verhältnis zwischen der größten und der kleinsten darstellbaren positiven Zahl.

Geben Sie für die unter a) und b) betrachteten 32-bit-Festpunkt- bzw. Gleitpunktzahlen den Dynamikbereich, jeweils in Potenzen von 2 und 10 (näherungsweise) an.

D_{fix} = ... = ... (2) = ... (10)

D_{float} = ... = ... (2) = ... (10)

d) Der Dynamik-Bereich wird typischerweise in Dezibel (dB) angegeben. Für eine Größe X berechnet sich dieser Wert X' zu:

$$X' = 20 \cdot \log_{10} X \quad (dB).$$

Geben Sie für die unter c) berechneten Werte D_{fix} und D_{float} die entsprechenden Werte D'_{fix} und D'_{float} in dB an:

D'_{fix} = ... dB

D'_{float} = ... dB

(Lösung auf Seite 189)

Aufgabe 46: Berechnung des Divisionsrests (I.3.2)

a) Geben Sie die Formel für die Berechnung des Restes der Division zweier Operanden A und B an. Geben Sie den Rumpf eines Assemblerprogramms zur Berechnung dieser Formel durch die anderen arithmetischen Operationen an.

b) Wie sieht das Ergebnis der Befehle FGETEXP und FGETMAN im 32-bit-Format nach dem IEEE-754-Standard aus, wenn man sie auf die folgende Dezimalzahl anwendet? $Z = -1{,}625 \cdot 2^{160}$.

(Lösung auf Seite 189)

Aufgabe 47: MMX-Rechenwerk (einfache Befehle) (I.3.2)

Ein MMX-Rechenwerk *(Multimedia Extension)* unterstützt gepackte Datenformate, die in einem 64-bit-Register wahlweise 8 Bytes, 4 (16-bit-)Wörter, 2 (32-bit-)Doppelwörter oder ein 64-bit-Wort (Quadword) unterbringen. Alle Werte können vorzeichenlos oder vorzeichenbehaftet sein. Negative Werte werden dabei im Zweierkomplement dargestellt. Spezielle MMX-Befehle wirken parallel auf diese Datenformate, d.h. es können z.B. durch einen einzigen Addier-Befehl zweimal 8 Bytes addiert werden. Ein Übertrag zwischen den einzelnen Werten der gepackten Daten findet dabei nicht statt. Die Datenbreite wird im Assemblerbefehl spezifiziert. Der erwähnte Addier-Befehl hat z.B. die Form: PADDX, wobei X = B, W, D für 8 Bytes, 4 Wörter bzw. 2 Doppelwörter steht.

a) Die Addition mit vorzeichenbehafteten Werten kann wahlweise in einer der beiden folgenden Formen geschehen:

- **mit Abschneiden** (Befehl PADD, *Wrap Around*):
  Ein (Teil-)Ergebnis, das nicht ins vorgegebene Datenformat paßt, verursacht einen Über- oder Unterlauf im darstellbaren Zahlenbereich und wird auf die durch die Datenlänge definierte Anzahl von (niederwertigen) Bits abgeschnitten (modulo-2^n-Rechnung, n = 8, 16, 32, 64).
- **mit Sättigung** (Befehl PADDS, PADDUS, *Saturation*):
  Das Ergebnis wird bei einem Überlauf auf den größten darstellbaren Wert aufgerundet bzw. bei einem Unterlauf auf den kleinsten Wert abgerundet. Dabei können die als Zahlen vorzeichenlos (U) oder vorzeichenbehaftet angesehen werden.

Geben Sie den Wert des Ergebnisregisters R1 an nach Ausführung der Befehle

1. PADDD R1, R2 (R1 := R1 + R2 mit *Wrap Around*)
2. PADDUSB R1, R2 (R1 := R1 + R2 vorzeichenlos, mit Sättigung)
3. PADDSW R1, R2 (R1 := R1 + R2 vorzeichenbehaftet, mit Sättigung)

mit der Eingabe

R1 = $ 6AE303D2835A7F34, R2 = $ 12F0A234AF061707.

(Diese Eingabe soll für jeden Befehl erneut verwendet werden.)

b) Geben Sie für den Eingabewert R1 = $8756 5432 F234 00F5 das Ergebnis des folgenden Befehls zum arithmetischen Rechtsschieben an:

PSRAW R1, #5 (Ergebnis in R1)

in dem #5 die Anzahl der Bitpositionen angibt, um die (parallel) verschoben werden soll.

(Lösung auf Seite 190)

Aufgabe 48: MMX-Rechenwerk (Multiplikation) (I.3.2)

Ein MMX-Rechenwerk *(Multimedia Extension)* unterstützt gepackte Datenformate nach untenstehendem Bild 14, die in einem 64-bit-Register wahlweise 8 Bytes, 4 (16-bit-)Wörter, 2 (32-bit-)Doppelwörter oder ein 64-bit-Wort (Quadword) unterbringen. Alle Werte können vorzeichenlos oder vorzeichenbehaftet sein. Negative Werte werden dabei im 2er-Komplement dargestellt. Spezielle MMX-Befehle wirken parallel auf diese Datenformate, d.h. es können z.B. durch einen einzigen Addier-Befehl zweimal 8 Bytes addiert werden. Ein Übertrag zwischen den einzelnen Werten der gepackten Daten findet dabei nicht statt. Die Datenbreite wird im Assemblerbefehl spezifiziert. Der erwähnte Addier-Befehl hat z.B. die Form: PADDX, wobei X = B, W, D für 8 Bytes, 4 Wörter, 2 Doppelwörter steht. Alle Befehle werden im Zweiadreß-Format benutzt, d.h. das Ergebnis wird in einem der Eingaberegister abgelegt.

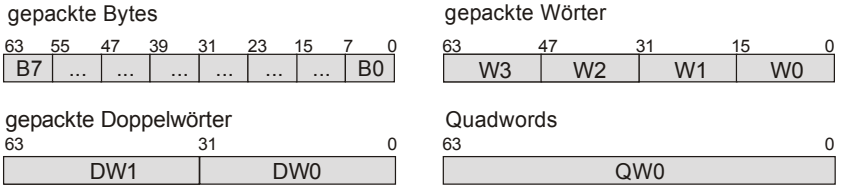


Bild 14: Datenformate des MMX-Rechenwerks

Der MMX-Befehlssatz enthält u.a. die in Tabelle 4 angegebenen Befehle.

Tabelle 4: MMX-Befehlssatz

Mnemo		Bemerkung
Logische Operationen (wirken auf alle 64 Bits)		
PAND	*Packed And*	bitweise Und-Verknüpfung
PANDN	*Packed And-Not*	bitweise Und-Verknüpfung mit negiertem erstem Operanden
POR	*Packed Or*	bitweise Oder-Verknüpfung
PXOR	*Packed Exclusive Or*	bitweise Antivalenz-Verknüpfung
Vergleichsbefehle (für X = B, W, D)		
PCMPEQX	*P. Compare Equal*	elementeweiser, vorzeichenloser Vergleich auf gleich
PCMPGTX	*P. Compare Greater*	elementeweiser, vorzeichenloser Vergleich auf größer
Transferbefehle		
MOVQ	*Move Quadword*	Transfer zw. MMX-Register und MMX-Register bzw. Speicher
MOVD	*Move Doubleword*	Transfer zw. MMX-Reg. und MMX-Reg./Speicher
PSWAPD	*P. Swap Doubleword*	Vertauschen der Doppelwörter in Registern, s. Skizze
Multiplizier/Addierbefehl (W × W → D)		
PMUL-ADDWD	*Multiply-Add*	4fache Multiplikation mit Addition zu Doppelwörtern
PADDX	*Packed Add*	Parallele Addition mit X = B, W, D

Als Ergebnis liefern die Vergleichsbefehle für jedes Element (B, W, D) den Wert $F...F (also eine Folge von ‚1'-Bits), wenn der Vergleich wahr ist, andernfalls den Wert $0...0 (also eine Folge von ‚0'-Bits). Dies ist in Bild 15 skizziert.

Die Befehle werden im Zweiadreß-Format angegeben. Dabei bedeutet die Assemblernotation <Befehl> R1, R2:

Der Inhalt von Register R1 wird mit dem Inhalt von R2 durch die Operation <op> verknüpft und das Ergebnis wird im Register R1 abgelegt.

In Kurzform: R1 := R1 <op> R2.

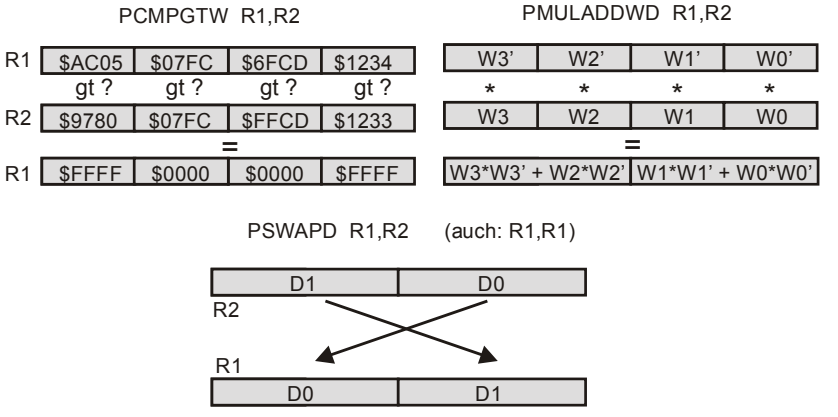


Bild 15: Skizze einiger MMX-Befehle

a) In den 64-bit-Registern R1, R2 seien die folgenden gepackten Wörter vorgegeben:

R1 = (W3, W2, W1, W0) und R2 = (W3', W2', W1', W0').

Geben Sie eine Befehlsfolge aus den beschriebenen Befehlen an, die im Ergebnisregister das wortweise Maximum der beiden Operandenregister R1, R2 enthält. Die Eingaberegister dürfen dabei überschrieben werden.

Führen Sie die Befehlsfolge an folgendem Beispiel durch:

R1 = ($C7A0, $50FE, $0800, $7F0B) und

R2 = ($9F40, $50FE, $A060, $3400).

b) Geben Sie an, wie man mit dem Befehl PMULADDWD „doppelt-genaue" Multiplikationen „$16 \times 16 \rightarrow 32$ bit" ausführen kann?

c) Im Speicher seien ab der Startadresse Adr_1 acht 16-bit-Wörter Wi, i = 0, ..., 7, und ab der Startadresse Adr_2 acht 16-bit-Koeffizienten Ci, i = 0, ..., 7 abgelegt. Für diese Zahlen soll der folgende Ausdruck berechnet :

$$S = \Sigma_{i=0,..,7}\ Ci \cdot Wi$$

Geben Sie eine MMX-Befehlsfolge für die geforderte Berechnung an.
Führen Sie die Befehlsfolge symbolisch mit den Bezeichnern Wi, Ci durch.

Hinweise

- Die Summe soll im niederwertigen Teil des Ergebnisregisters stehen; der Wert des höherwertigen Teils im Register sei irrelevant.
- Die Startadressen der Operanden Wi, Ci für i = 4, ..., 7 seien Adr_1+4 und Adr_2+4.

(Lösung auf Seite 191)

Aufgabe 49: MMX-Rechenwerk (Bildbearbeitung) (I.3.2)

In dieser Aufgabe werde wiederum das MMX-Rechenwerk *(Multimedia Extension)* betrachtet, das in Aufgabe 48 beschrieben wurde. In Tabelle 4 finden Sie dort einen Ausschnitt des MMX-Befehlssatzes, im Bild 14 werden die unterstützten Datenformate dargestellt und im Bild 15 wird die Ausführung wichtiger Befehle skizziert.

a) In vier Bereichen eines Graphikspeichers seien die im folgenden Bild 16 dargestellten 8×8-byte-Ausschnitte zweier Bildschirmfenster (Bereiche 2, 1), eine gleich große Maske (Bereich 0) sowie ein Ausgabefeld (Bereich 3) abgelegt. Die Adressierung der korrespondierende Zeilen dieser Bereiche geschieht – wie im Bild gezeigt – jeweils Register-indirekt über ein Adreßregister I3, ..., I0 mit der Assemblerdarstellung: (Ij). Durch die benutzten Farben der Elemente weiß, grau, schwarz sollen die Hexadezimalwerte $00, $80, $FF repräsentiert werden.

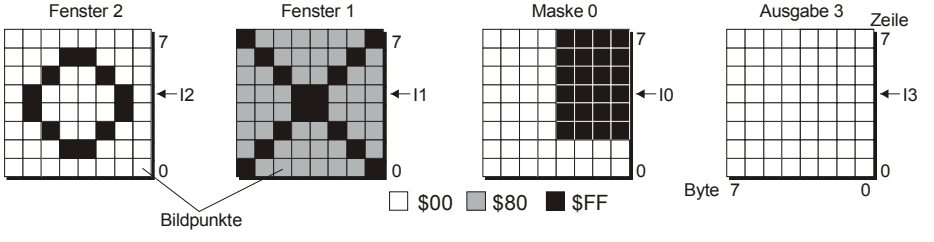


Bild 16: Ausschnitte des Graphikspeichers

Geben Sie eine Befehlsfolge an, die

- die Zeilen der Bereiche i in die Register Ri (i = 0, 1, 2) einliest,
- die Register R2, R1 mit dem Register R0 nach folgender Vorschrift parallel zum Ergebnisregister R3 verknüpft (j = 0, ..., 7):

  Byte j von R3 = Byte j von R2, falls Byte j von R0 = $00

  Byte j von R3 = Byte j von R1, falls Byte j von R0 = $FF.
- das Ergebnisregister in den Ausgabebereich 3 überträgt.

Skizzieren Sie durch verschiedene Muster (weiß, schwarz, schraffiert) im Ausgabebereich 3 des Bildes das Ergebnis der oben stehenden Befehlsfolge für alle Zeilen.

b) Die Byte-Vektoren im folgenden Bild 17 repräsentieren jeweils acht Punkte, die auf einer Linie zweier dreidimensionaler Objekte liegen und sich bei der Bildschirmdarstellung überlagern. Die oberen Zahlen geben dabei die Farbwerte der Punkte wieder, die unteren ihren Abstand von der Bildschirmoberfläche: je größer dieser Wert, desto „tiefer“ liegt der Punkt.

Geben Sie eine Befehlsfolge an, die parallel für alle Paare der Punktvektoren diejenigen Punkte ermittelt, die näher an der Bildschirmoberfläche liegen, und die Farb- und Tiefenwerte dieser Punkte in die Ausgabevektoren überträgt. Es werde dabei davon ausgegangen, daß alle Byte-Vektoren im Speicher abgelegt sind.

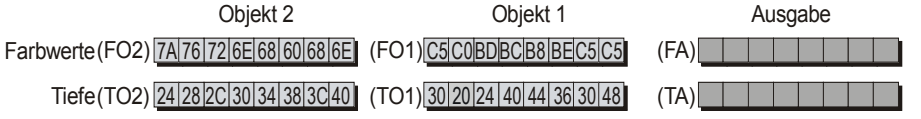


Bild 17: 8 Punkte des Bildschirms

Benutzen Sie für Ihre Lösung bitte die im Bild in Klammern angegebenen Variablennamen. Tragen Sie die Ergebnisse der Befehlsfolge in die Ausgabevektoren FA, TA ein.

(Lösung auf Seite 192)

Aufgabe 50: MMX-Rechenwerk (Minimum, Maximum) (I.3.2)

In dieser Aufgabe werde wiederum das MMX-Rechenwerk *(Multimedia Extension)* betrachtet, das in Aufgabe 48 beschrieben wurde. In Tabelle 4 finden Sie dort einen Ausschnitt des MMX-Befehlssatzes, im Bild 15 und wird die Ausführung wichtiger Befehle skizziert und im Bild 14 werden die unterstützten Datenformate dargestellt.

Geben Sie für die Operandenregister R0, R1 mit jeweils vier 16-bit-Wörtern eine Folge von logischen Befehlen und MOV-Befehlen an, durch die in einem Ergebnisregister Ri

- in seinen beiden höherwertigen („linken“) 16-bit-Wörtern die Maxima,
- in seinen beiden niederwertigen („rechten“) 16-bit-Wörtern die Minima

von R0 und R1 abgelegt werden, wobei Maxima und Minima elementeweise über R0, R1 ermittelt werden.

Beispiel

R0 = ($073F,$A0A0,$6324,$370A), R1 = ($A37F,$827A,$2F3E,$8520)
ergeben: R2 = ($A37F,$A0A0,$2F3E,$370A)

Nr.	Befehl	Bemerkung
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

(Lösung auf Seite 194)

Aufgabe 51: Befehlsbearbeitungszeiten (I.3.2)

Für zwei Prozessoren A, B, die denselben Befehlssatz verarbeiten, gelten die folgenden Annahmen:

- Eine Messung von typischen Programmen für die Prozessoren A, B ergab die in der Tabelle aufgeführten Auftrittshäufigkeiten der wichtigsten Befehle:

Befehl	LOAD	STORE	ADD	SUB	COMPARE	BRANCH	Rest
Häufigkeit	22 %	12 %	8 %	5 %	16 %	20 %	17 %

- Prozessor B braucht für einen Speicherzugriff nur halb so viele Taktzyklen wie Prozessor A, für alle anderen Befehle jedoch genau so viele wie A.
- Prozessor A benötigt für einen Speicherzugriff 4 Taktzyklen, für jeden anderen Befehl nur einen Takt.
- Die Taktrate von Prozessor B ist um 15 % kleiner als die von Prozessor A.

a) Geben Sie für beide Prozessoren die durchschnittliche Befehlsbearbeitungszeiten für den oben dargestellten Befehlsmix (in Taktzyklen pro Befehl) an.

b) Welcher Prozessor bearbeitet die zugrunde gelegten typischen Programme schneller? Geben Sie die Berechnung an!

c) Um wie viel Prozent ist er schneller? Geben Sie die Berechnung an!

(Lösung auf Seite 195)

Aufgabe 52: Befehlssatz und Adressierungsarten (I.3.3)

In dieser Aufgabe sollen Sie das folgende kleine Assemblerprogramm analysieren und hinsichtlich der Programmlänge optimieren. Der 16-bit-Prozessor habe einen 16-bit- Adreßbus. Die Darstellung im Speicher erfolgt im *Big-Endian*-Format. Die Assemblerbefehle seien mit den in Abschnitt I.3.2 beschriebenen identisch und im Zweiadreß-Format gegeben:

<Befehl> Ri, Rj entspricht: Ri := Ri <op>Rj.

Label	Befehl		Adressierungsart
Start:	ROL	$56	
	EOR	$02(R4), R3	
	DEC	R1	
	BNE	Start	
	ADD	R3, ($006E)	
	ADD	R2, R3	
	ST	$0061, R2	

a) Ergänzen Sie die obige Tabelle um die bei den Befehlen verwendeten Adressierungsarten.

b) Vor der Ausführung des obigen Programms haben der Datenspeicher und die Register den Inhalt:

Speicherauszug (alle Werte in hexadezimaler Form):

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
005X	92	61	5D	2C	94	88	00	0A	60	02	22	19	87	AF	D0	11
006X	71	34	8D	D4	23	55	96	A0	00	57	10	22	94	AE	00	56

Registerbelegung:

R1 = $0010 R2 = $0123 R3 = $0001 R4 =$0054

Welchen Inhalt haben Register und Speicher nach der Ausführung des oben stehenden Programms? (Es reicht, nur die geänderten Zellen einzutragen.)

Register:

R1 = $..... R2 = $..... R3 = $..... R4 = $.....

Speicher:

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0050																
0060																

c) Welche Aufgabe erfüllt dieses Programm?

d) Wie läßt sich diese Aufgabe unter Verwendung der im Kapitel I.3 beschriebenen Befehle und Adressierungsarten mit minimalem Aufwand realisieren?

(Lösung auf Seite 195)

Aufgabe 53: Befehlssatz und Adressierungsarten (I.3.3)

Ein 8-bit-Mikroprozessor besitze einen Akkumulator A, ein 8-bit-Indexregister IR sowie ein 16-bit-Basisregister BR. BR und IR können automatisch dekrementiert bzw. inkrementiert werden.

In der folgenden Tabelle 5 ist die Anfangsbelegung dieser Register sowie ein Ausschnitt aus der Belegung des Arbeitsspeichers dargestellt. Dieser Ausschnitt ist in der üblichen Matrixform gegeben. Beginnend mit der Basisadresse $A900, sind die Speicherzellen nach wachsenden Adressen zeilenweise von links nach rechts und von oben nach unten angeordnet.

(Beispiel: Die Adresse der unterstrichenen Speicherzelle mit dem Inhalt $86 wird durch Addition der hexadezimalen Spaltennummer $B zur links vor der Zeile stehenden Anfangsadresse $A960 gewonnen: $A960 + $B = $A96B.)

Beginnend mit dem durch die Tabelle gegebenen Anfangszustand, führe nun der Prozessor die folgende Sequenz von Assemblerbefehlen aus. Die Notation der Adressierungsarten stimmt dabei mit der im Abschnitt I.3.3 benutzten überein.

Bei der Befehlsausführung ist zu beachten, daß (16-bit-)Adressen so im Speicher abgelegt werden, daß das höherwertige Byte unter der niedrigeren Speicheradresse liegt (*Big-Endian*-Format).

Tabelle 5: Anfangsbelegung der Register und Speicherbelegung

Register: A: $00 ; IR: $F5 ; BR: $A900 ⇓

Adresse	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
A900	A9	F1	10	8E	00	06	C6	5A	BD	F1	20	8E	00	02	BD	F1	
A910	56	10	9F	00	8E	00	06	C6	EA	BD	F1	20	8E	00	02	BD	
A920	F1	56	BD	F1	10	1F	20	93	00	2A	06	81	FF	27	0D	20	
A930	03	4D	27	08	BD	F1	23	CC	38	7C	20	06	BD	F1	20	CC	
A940	6D	7C	8E	00	06	BD	F1	16	BD	F1	43	C1	80	27	B1	C1	
A950	81	26	F5	8E	00	A9	DC	00	BD	F1	23	8E	00	00	1F	20	
A960	BD	F1	23	20	E3	BD	F1	10	8E	00	05	**86**	02	BD	F1	43	⇐
A970	A9	0A	24	F9	E7	82	BD	F1	1C	4A	2A	F1	8E	00	00	96	
A980	A9	C1	01	22	3A	C6	64	3D	D7	00	96	03	C6	0A	3D	DB	
A990	00	DB	04	D7	00	C1	80	24	26	BD	F1	43	C1	86	27	C5	
A9A0	C1	80	26	0E	D6	00	86	70	BD	F1	20	30	07	BD	F1	13	
A9B0	20	E7	C1	81	26	E3	D6	00	C8	FF	5C	86	40	20	E9	10	
A9C0	6D	7C	8E	00	06	BD	F1	16	56	10	9F	00	8E	00	06	C6	
A9D0	BD	F1	10	8E	00	06	C6	5A	26	BD	F1	43	C1	86	27	C5	
A9E0	C1	80	26	0E	D6	00	86	70	BD	F1	23	8E	00	00	1F	20	
A9F0	81	26	F5	8E	00	A9	30	00	C8	FF	5C	86	40	20	E9	10	

1) INC ($A955) erhöhe den Inhalt der adressierten Speicherzelle um 1

2) CLR ((BR)(IR)+) setze die adressierte Speicherzelle auf den Wert $00 zurück

3) LDA #$7F lade den Akkumulator A mit $7F
   STA $98(BR) speichere A unter der effektiven Adresse

4) LD IR,#$27 lade das Indexregister IR mit $27
   LDA #$3F lade den Akkumulator A mit $3F
   STA $A900(IR) speichere A unter der effektiven Adresse

5)	LDA	(BR)(IR)	lade den Akkumulator mit dem Inhalt der adressierten Speicherzelle
6)	STA	((BR))	speichere A unter der effektiven Adresse
7)	LD	IR,#$60	lade das Indexregister IR mit $60
	LDA	#$11	lade den Akkumulator A mit $11
	ADD	$A9B0	addiere zum Inhalt von A den Inhalt der adressierten Speicherzelle
	STA	(BR)+	speichere A unter der effektiven Adresse
8)	ST	BR,$A900(IR)	speichere den aktuellen Inhalt von BR unter der effektiven Adresse
9)	CLR	$10($6F(BR))(IR)	setze die adressierte Speicherzelle auf den Wert $00 zurück

Tragen Sie in die oben stehende den Inhalt aller Speicherzellen ein, die durch einen der oben aufgeführten Befehle manipuliert wurden. Schreiben Sie an jede geänderte Speicherzelle als Index die Nummer 1) – 9) der Befehlsgruppe, durch die die Änderung verursacht wurde.

Als Beispiel ist in der Tabelle bereits die Wirkung des folgenden Befehls eingetragen worden:

0)	NOT $D8(BR)	Invertiere bitweise den Inhalt der adressierten Speicherzelle

effektive Adresse: EA = (BR) + ((PC) + 1) = $A900 + $D8 = $A9D8

Inhalt der adressierten Speicherzelle $A9D8: $26

$26 = 0010 0110 —NOT→ 1101 1001 = $D9.

(Lösung auf Seite 196)

Aufgabe 54: Zero-Page-Adressierung (I.3.3)

Der 8-bit-Mikroprozessor 6502 der Firma *MOS Technologies* aus der Mitte der 70er Jahre wurde millionenfach in den PET- und C64-Rechnern der Firma Commodore eingesetzt. Er besaß einen 8-bit-Datenbus und einen 16-bit-Adreßbus. 16-bit-Adressen wurden byteweise unter zwei hintereinander folgenden Adressen im Speicher abgelegt, wobei das höherwertige Adreßbyte unter der höherwertigen Adresse zu finden war (*Little-Endian*-Format). Der Prozessor enthielt unter anderem einen 8-bit-Akkumulator A und ein gleich langes Indexregister Y.

a) Geben Sie für die Adressierungsart:

„postindizierte indirekte Zero-Page-Adressierung“

mit der *Assemblerschreibweise*:

(<L-Adresse>),Y

die Berechnungsvorschrift für die effektive Adresse an und stellen Sie sie symbolisch (wie in den Bildern des Abschnitts I.3.3) dar.
In Maschinensprache umfaßt jeder Befehl mit dieser Adressierungsart zwei Bytes:

<OpCode> <L-Adresse>.

b) Der Befehlssatz dieses Prozessors enthielt u.a. die Befehle:

CLC	Lösche das Übertragsbit	*(Clear Carry Flag)*
LDA	Lade den Akkumulator A	*(Load Accu A)*
STA	Speichere den Akkumulator A	*(Store Accu A)*
ADC	Addiere den Speicheroperanden zum Akkumulator A unter Berücksichtigung des Übertrags	*(Add with Carry)*
LDY	Lade das Y-Register	*(Load Y)*
DEY	Dekrementiere den Inhalt des Y-Registers um 1	*(Decrement Y)*
BNE	Verzweige, falls *Zero Flag* = 0, (das *Zero Flag* wird als Ergebnis einer Operation genau dann gesetzt, wenn das A- bzw. Y-Register danach den Wert $00 hat)	*(Branch not Equal)*
JSR	Unterprogrammaufruf	*(Jump to Subroutine)*
RTS	Rücksprung aus Unterprogramm	*(Return from Subroutine)*

Schreiben Sie ein Unterprogramm SUM, das im Festwertspeicher abgelegt werden kann und die 16-bit-Summe über einen maximal 255 byte langen Datenbereich berechnet. Die Anfangsadresse von SUM sei $E000. Die um ‚1' dekrementierte Anfangsadresse des Datenbereiches werde dem Unterprogramm in den Speicherzellen $00A0 und $00A1, die Länge des Datenbereiches (ungleich 0) im Y-Register übergeben. Das Endergebnis werde in den Speicherzellen $00A2 und $00A3 zum Hauptprogramm übertragen. Stellen Sie den Unterprogrammaufruf dar, falls die Summe der 128 Bytes ab der Basisadresse $1000 berechnet werden soll.

(Lösung auf Seite 197)

Aufgabe 55: Indirekte Adressierung (I.3.3)

Ein 8-bit-Mikroprozessor besitze einen 8-bit-Datenbus und einen 16-bit-Adreßbus. 16-bit-Adressen werden byteweise unter zwei hintereinander folgenden Adressen des Speichers abgelegt, wobei das höherwertige Adreßbyte unter der höherwertigen Adresse zu finden ist (*Little-Endian*-Format). Der Prozessor besitzt unter anderem einen 8-bit-Akkumulartor A und ein 16 bit langes Indexregister Y.

a) Beschreiben Sie ausführlich einen Ladebefehl für den Akkumulator A für die Adressierungsart „indirekte Register-indirekte Adressierung" LDA ((Y)).

b) Es liege der in Tabelle 6 vorgegebene Speicherauszug vor.

Tabelle 6: Ausschnitt der Speicherbelegung

Adresse	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0390	0F	04	0B	04	0A	04	08	04	06	04	04	04	02	04	00	04
0400	10	04	90	03	12	04	92	03	14	04	94	03	16	04	06	03
0410	03	04	01	40	93	03	92	03	95	03	07	04	17	04	1F	04

Welche Werte werden bei der obigen Adressierungsart durch ‚LDA ((Y))' in den Akkumulator geladen, wenn das Indexregister Y die in der folgenden Tabelle 7 gegebenen Werte besitzt? Tragen Sie die effektive Adresse und den Akkumulatorwert in diese Tabelle ein.

Tabelle 7: Effektive Adressen und Akkumulatorwerte

Indexregister Y	effektive Adresse	Akkumulator A
0404		
041e		
0399		
040d		

c) Schätzen Sie ab, wie viele Takte der obige Ladebefehl benötigt. Die Länge des Befehls soll 2 Byte betragen. Jeder Zugriff auf den Speicher benötige einen Taktzyklus.

(Lösung auf Seite 198)

Aufgabe 56: Adressierungsarten (Umwandlung CISC → RISC) (I.3.3)

Ein einfacher RISC-Prozessor besitze nur die in der folgenden Tabelle 8 angegebenen sechs verschiedenen Adressierungsarten, von denen für arithmetische, logische und Verschiebebefehle nur die erste benutzt wird. Für diese Befehle müssen alle Operanden in den Registern des Prozessors vorliegen. Die letzten fünf Adressierungsarten dienen nur in den Befehlen LOAD und STORE zur Selektion einer Speicherzelle sowie in Sprung- und Verzweigungsbefehlen zur Angabe des Sprungzieles. (EA: effektive Adresse).

Der Prozessor verfüge weiterhin über einen Satz von 32 universellen 32-bit-Registern R31, ..., R0, die sowohl als Daten- wie Adreßregister benutzt werden können. Es stehen alle im Kurs beschriebenen Befehle zur Verfügung. Der Prozessor unterstütze nur das Zweiadreß-Format für Operanden, die in Registern zur Verfügung stehen. Dabei entspreche dem Befehl

<Operation> Ri, Rj

die Zuweisung Ri := Ri <op> Rj.

Tabelle 8: Adressierungsarten des RISC-Prozessors

Bezeichnung	Notation	Effektive Adresse
für arithmetische, logische, Verschiebebefehle und Transferbefehle		
Explizite Register-Adressierung	Ri	EA = i
zusätzlich für LOAD/STORE-Befehle		
Unmittelbare Adressierung (nur *Load*)	#<Operand>	EA = (PC)
Register indirekte Adressierung	(Ri)	EA = (Ri) (Inhalt von Ri)
Register-relative Adressierung:	<Offset>(Ri)	EA = (Ri) + <Offset>
Register-relative Adressierung mit Index	(Ri)(Rj)	EA = (Ri) + (Rj).
Programmzähler-relative Adressierung	<Offset>(PC)	EA = (PC) + <Offset>

a) Geben Sie für die Nachbildung der im Abschnitt I.3.3 beschriebenen komplexen CISC-Adressierungsarten jeweils eine RISC-Befehlsfolge für den Lesebefehl

LD R0,<Operand>

an. Kommentieren Sie jeden RISC-Befehl. Versuchen Sie, mit möglichst wenigen Registern Ri auszukommen.

1. Absolute Adressierung: LD R0,<Adresse>
2. Indirekte absolute Adressierung: LD R0,(<Adresse>)
3. Indirekte, indizierte Adressierung: LD R0,(<Offset>(R1)(R2))
4. Indizierte, indirekte Adressierung: LD R0, <2.Offset>(<1.Offset>(R1))(R2)

b) Der RISC-Prozessor verfüge über keine speziellen Flag-Manipulationsbefehle. Geben Sie Befehlsfolgen an, mit der in einem Register Rj das Bit i gezielt gesetzt bzw. zurückgesetzt werden kann. Kommentieren Sie die Befehle.

1. Setzen: Vorgaben: Rj, j = 0; Bit i, i = 15
2. Zurücksetzen: Vorgaben: Rj, j = 0; Bit i, i = 22

(Lösung auf Seite 199)

Aufgabe 57: Befehlsanalyse (OpCode) (I.3.2 – I.3.3)

Gegeben sei ein Mikroprozessor, der Zugriff auf zwei Indexregister (B0 und B1) hat, die jeweils eine Adresse im Hauptspeicher beinhalten können. Weiterhin existieren zwei Datenregister (A und C), die jeweils ein 16-bit-Wort speichern können. Im Assemblercode (s.u.) werden diese Register durch die Bezeichner R („Register“) und I („Indexregister“) repräsentiert und im OpCode durch die in der Tabelle 9 angegebene Bitfolge selektiert.

Tabelle 9: Definition der Register

	R				I	
Register	A	C	B0	B1	B0	B1
Bitfolge	00	01	10	11	0	1

Der Befehlssatz des Prozessors bestehe aus den in der Tabelle 10 gegebenen 16 bit langen Befehlen. Bei der darin angegebenen Adresse T handelt es sich um eine 13-bit-Speicheradresse, der Operand K stellt eine 11-bit-Konstante dar.

Tabelle 10: Befehlstabelle

OpCode	Befehl	Oper.	Funktion
000	JMP	T	springe zur Adresse T
001	ADD	R, K	addiere die Konstante K zu R und speichere das Ergebnis in R
010	SKIP	R, K	Überspringe den nächsten Befehl, wenn der Inhalt von R gleich der Konstanten K ist
011	LOADK	R, K	lade R mit der Konstanten K
100	LOADI	R, I	lade R mit dem Inhalt der Speicherzelle, deren Adresse in I steht
101	STORE	T, R	speichere R in die Speicherzelle T
110	STOREI	I, R	speichere R in die Speicherzelle, deren Adresse in I steht
111	LOAD	R, T	lade R mit dem Inhalt der Speicherzelle T

a) Gegeben sei nun das folgende Assemblerprogramm, welches auf dem Prozessor ausgeführt wird.

Zeile	Marke	Befehl	Argumente
1		LOADK	C,#0
2	L1:	LOADI	A, B0
3		STOREI	B1, A
4		SKIP	A,#0
5		JMP	END
6		ADD	C,#1
7		ADD	B0,#2
8		ADD	B1,#2
9		JMP	L1
10	END:		

Ergänzen Sie die folgende Tabelle, indem Sie zu jeder Programmanweisung die zugehörigen Werte in binärer Form eintragen.

Zeile	OpCode Register	Operand
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

b) Ergänzen Sie die folgende Tabelle, indem Sie zu jeder Anweisung den zugehörigen Ausdruck in der Hardware-Beschreibungsnotation angeben, wie sie auch im Kapitel I.3 verwendet wird.

Zeile	Operation
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

c) Ergänzen Sie den folgenden Satz:

Es handelt sich um eine .....–Adreßmaschine.

d) Beschreiben Sie kurz die Funktion des Programms und den Inhalt von C.

(Lösung auf Seite 200)

Aufgabe 58: Programmanalyse (I.3.2 – I.3.3)

Gegeben sei das in Tabelle 11 stehende Programmsegment für einen einfachen Mikroprozessor. Alle Register des Prozessors seien 16 bit breit. Dyadische Operationen werden im Zweiadreß-Format in der folgenden Form angegeben:

<Operation> R1,R2 entspricht: R1 := R1 <Operation> R2

Dabei kann der zweite Operand auch ein unmittelbar *(immediate)* eingegebener Wert sein. ADR bezeichnet eine beliebige, aber feste Speicheradresse; der Wert der adressierten Speicherzelle sei $C7AB. Die Mnemocodes der Befehle und ihre Bedeutungen stimmen mit den in Abschnitt I.3.2 beschriebenen Befehlen überein. Von den aufgeführten Befehlen soll nur der RCL-Befehl *(Rotate with Carry Left)* sich auf das *Carry Flag* auswirken.

Tabelle 11: Ein kleines Programmsegment

Nr.	Befehl			Bedeutung
1		LD	R0,#$0010	
2		LD	R1,(ADR)	
3		CLR	R2	
4	L:	RCL	R1	
5		BCC	M	
6		INC	R2	
7	M:	DEC	R0	
8		BNE	L	
9		RCL	R1	
10		AND	R2,#$0001	
11		ST	(ADR+1),R2	

a) Tragen Sie in Tabelle 11 die Bedeutung der einzelnen Befehle ein!

b) Geben Sie allgemein und für die o.g. Anfangsbelegung der durch ADR adressierten Speicherzelle an, welche Werte die Register R1 und R2 nach der Ausführung der 9. Programmzeile besitzen. Welche „Aufgabe“ hat der 9. Befehl?

c) Geben Sie allgemein und für die o.g. Anfangsbelegung der durch ADR adressierten Speicherzelle an, welche Werte die Speicherzellen mit den Adressen ADR und ADR+1 nach der Ausführung der 11. Programmzeile besitzen.

d) Der JMP-Befehl *(Jump)* des unter a) beschriebenen Prozessors erlaube nur die absolute Adressierung des Sprungziels. Geben Sie eine kurze Befehlsfolge an, die es gestattet, „Register-indirekt zu springen“, d.h. zu einer Adresse, die in einem Register R vorgegeben wird. (Dazu dürfen alle in Abschnitt I.3.2 beschriebenen Befehle benutzt werden. Jedoch sei ein direktes Laden des Programmzählers aus einem Register nicht möglich.)

(Lösung auf Seite 201)

I.4 Moderne Hochleistungsprozessoren

Aufgabe 59: DRAM/Cache-Zugriffszeiten (I.4.2)

Ein Arbeitsspeicher bestehe aus DRAM-Bausteinen mit einer Zugriffszeit von 50 ns, der vorgeschaltete Cache aus SRAM-Bausteinen mit 10-ns-Zugriffszeit. In dieser Zeit sei die *Hit/Miss*-Entscheidung enthalten. Die Buszykluszeit des Prozessors sei 10 ns. Der Prozessor soll auf den Cache ohne Wartezyklen, auf den Arbeitsspeicher mit vier Wartezyklen zugreifen können. Die „Trefferrate“ der Cache-Zugriffe betrage 80 %. Berechnen Sie die durchschnittliche Zugriffszeit auf ein Datum sowie die durchschnittliche Anzahl der benötigten Wartezyklen.

(Lösung auf Seite 202)

Aufgabe 60: Cache (Trefferrate im MC68020) (I.4.2)

Nach der Fallstudie im Unterabschnitt I.4.2.3 besaß der Motorola MC68020 als einer der ersten Mikroprozessoren einen integrierten Cache mit einer Kapazität von 256 byte. Berechnen Sie die Trefferrate im Cache des MC68020, wenn dieser ununterbrochen eine Programmschleife abarbeitet, die aus genau 512 Bytes besteht.

(Lösung auf Seite 202)

Aufgabe 61: Cache-Organisation (Direct Mapped Cache) (I.4.2)

Das folgende Bild 18 zeigt den Aufbau eines direkt abbilden Caches *(Direct Mapped Cache)* mit einem Ausschnitt seiner aktuellen Belegung.

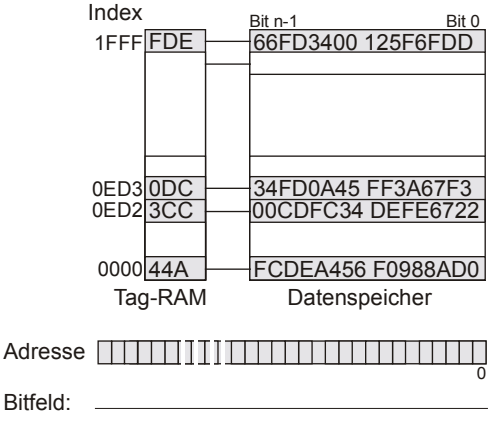


Bild 18: Belegung eines *Direct-Mapped Caches*