Aufgabe 1 (10 Punkte)

Geben Sie für die folgenden Aussagen an, ob sie richtig oder falsch sind.

a) Unter Hardware-Interlocking versteht man eine Maßnahme, den Systembus gegen unerlaubte Zugriffe von externen Rechnerkomponenten zu sperren.

☐ richtig ☒ falsch

Hardware-Interlocking ist eine Maßnahme zur Behebung von Pipelinekonflikten.

b) Bei RISC-Prozessoren werden neben dem Befehlsumfang auch die möglichen Adressierungsarten eingeschränkt.

☒ richtig ☐ falsch

RISC-Prozessoren werden auch als LOAD-/STORE-Architekturen bezeichnet. Durch den Wegfall komplexer Adressierungsarten wird die Pipelining-Auslastung optimiert.

c) Unter dem Begriff DMA (Direct Memory Access) versteht man den direkten Zugriff des Prozessors auf den Arbeitsspeicher ohne Umweg über den Cache.

☐ richtig ☒ falsch

DMA bezeichnet den Transport von Daten vom/zum Arbeitsspeicher ohne Beteiligung des Prozessors.

d) Wenn die Technik der ungeordneten Befehlsausführung angewandt wird, können superskalare Prozessoren stets pro Taktzyklus zwei oder mehrere nützliche Befehle eines Programms beenden.

☐ richtig ☒ falsch

Durch ungeordnete Befehlsausführung wird eine gleichmäßige Auslastung der Funktionseinheiten angestrebt. Es gibt aber keine Garantie dafür, dass immer mindestens zwei Funktionseinheiten belegt werden können.

e) Das Ausgangssignal eines Digital/Analog-Wandlers kann unendlich viele verschiedene (analoge) Spannungswerte annehmen.

☐ richtig ☒ falsch

Mit einem n-Bit langen Wert können nur 2^n verschiedene Spannungswerte codiert werden. Der kleinste Abstand zwischen zwei Ausgangsspannungen ist daher auf $U_{ref} / 2^n$ beschränkt.

Aufgabe 2 (16 Punkte)

Beantworten Sie folgende Fragen:

a) Unter einem „Trap" versteht man

eine Ausnahmesituation, die durch bestimmte prozessorinterne Ereignisse verursacht wird und synchron zum Programmablauf auftritt. Dabei kann es sich um einen Fehler oder anomalen Prozessorzustand im Zusammenhang mit einem Befehl handeln.

(2)

b) Die wesentlichen Nachteile des Rückschreibverfahrens gegenüber dem Durchschreibverfahren bestehen darin, dass

der Erhalt der Cache-Kohärenz schwieriger zu gewährleisten ist und mehr Aufwand für die Cache-Steuerung erforderlich wird.

(2)

c) Benennen Sie die Stufen einer (Befehls)Pipeline eines (skalaren) RISC-Prozessors. Wie groß ist die maximal mögliche Steigerung des Durchsatzes?

Auch wenn die Befehlspipelines der einzelnen Prozessormodelle sehr unterschiedlich ausfallen, können stets die folgenden Stufen bei der Befehlsbearbeitung unterschieden werden:

- Holphase (Instruction Fetch Phase)
- Decodierphase (Decode Phase)
- Ausführungsphase (Execution Phase)
- Rückschreibphase

Durch den Einsatz einer n-stufigen Pipeline kann der Durchsatz des Prozessors maximal um den Faktor n gesteigert werden.

(3)

d) Nennen Sie mindestens zwei Pipelinehemmnisse und beschreiben Sie Möglichkeiten zu deren Lösung.

- Betriebsmittelabhängigkeit, z.B.: Wartezeiten beim Speicherzugriff

Durch das vorübergehende Anhalten der Pipeline (Hardware Interlocking) kann der Prozessor auf langsamere oder in Gebrauch befindliche Ressourcen warten. Durch die Verwendung unterschiedlich langer Pipelines entstehende Registernamensabhängigkeiten können zudem mittels Scoreboards und Register Renaming aufgelöst werden.

- Datenabhängigkeit, z.B.: Ein Operationsergebnis wird vom Folgebefehl benötigt

Auch den auftretenden Datenabhängigkeiten kann mittels Hardware Interlocking begegnet werden. Alternative Strategien zur Lösung von Datenabhängigkeiten sind

das durch einen Bypass Bus realisierte „Load Forwarding" und das Einfügen von NOP-Befehlen in den Programmcode.

- Kontrollflussabhängigkeit

Kontrollflußabhängigkeiten können durch Pipeline Flushing, Delayed Branch und Out-of-order-Execution gelöst werden.
Bei dem als „Pipeline Flushing" bezeichneten Verfahren werden alle noch in der Pipeline stehenden Befehle bei der Ausführung des Verzweigungsbefehls aus der Pipeline entfernt. Beim „Delayed Branch"-Vefahren werden nach jedem Sprungbefehl mehrere NOP-Befehle in den Programmtext integriert. Im Unterschied hierzu wird bei der „Out-of-order"-Ausführung versucht, die vorhandenen Befehle ergebnisneutral derart umzustellen, daß die nach dem Sprungbefehl stehenden Befehle auf jeden Fall mit ausgeführt werden müssen.

(3)

e) Ein direct-mapped Cache hat gegenüber einem vollassoziativen Cache den

Vorteil:

Der Komparator eines direct-mapped Caches kann wesentlich einfacher als der Komparator eines vollassoziativen Caches gehalten werden. Als Folge hiervon können deutlich größere Cache-Speicher realisiert werden.

Nachteil:

Jede Speicherstelle kann nur auf genau eine Stelle im Cache abgebildet werden, wodurch die eingelagerten Daten im Durchschnitt wieder schneller verdrängt werden. Als Folge kann es zum Flattern der Cache-Einträge kommen.

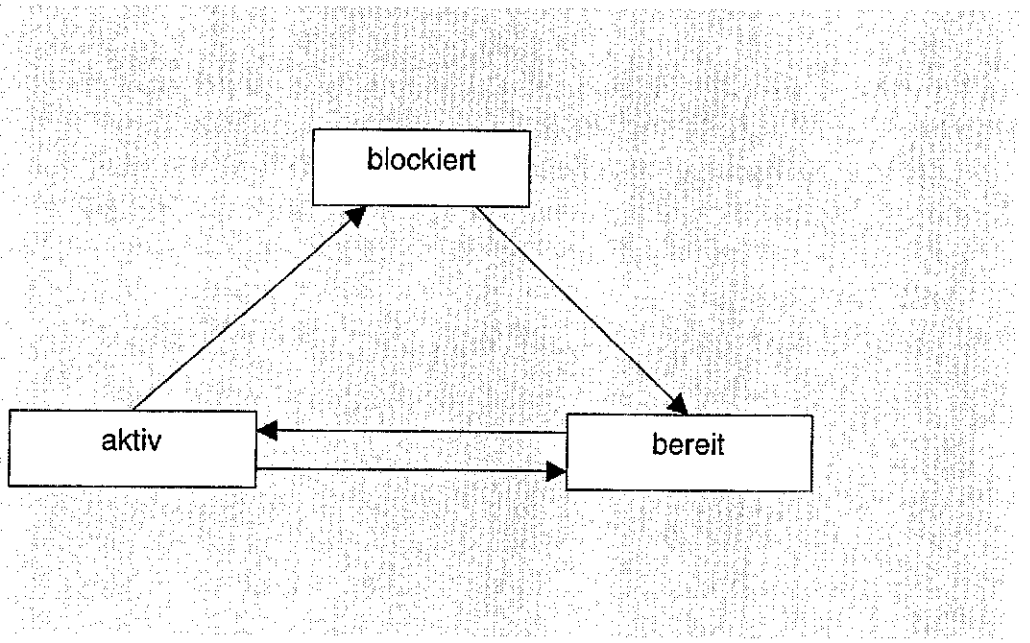
(3)

f) In welchen Zuständen kann sich ein Prozeß befinden und welche Zustandsübergänge gibt es? Skizzieren Sie ein Diagramm mit gerichteten Übergängen!

Man unterscheidet drei Zustände:

1. aktiv (running) : Der Prozeß wird gerade vom Prozessor bearbeitet;
2. blockiert (suspendet) : Der Prozeß muß auf ein bestimmtes Ereignis warten, z.B:
   - auf das Ende einer Ein-/Ausgabeoperation,
   - auf die Einlagerung von Daten in den Hauptspeicher,
   - auf einen anderen Prozeß;
3. bereit (ready) : Der Prozeß ist bereit, vom Prozessor ausgeführt zu werden, und muß insbesondere auf kein Ereignis warten.

Skizze:



(3)

Aufgabe 3 Pipelining (12 Punkte)

a) Bei einer fünfstufigen Pipeline eines (skalaren) RISC-Prozessors bewirken Sprungbefehle und durchgeführte bedingte Verzweigungsbefehle, dass die Pipeline für drei Taktzyklen angehalten werden muss. Um wieviel Prozent erhöht sich die durchschnittliche Dauer pro Befehl, wenn ein Testprogramm zu 20% aus den o.g. Befehlen besteht und wenn man annimmt, dass es daneben zu keinen weiteren Pipelinehemmnissen kommt?

Sofern kein Konflikt auftritt, wird nur ein Fetch-Zyklus pro Befehl benötigt. In den verbleibenden 20% der Fälle werden drei zusätzliche Zyklen benötigt. Hieraus folgt unmittelbar:

$$0{,}8 * 1 \text{ Zyklus} + 0{,}2 * 4 \text{ Zyklen} = 1{,}6 \text{ Zyklen (pro Befehl)}$$

Die durchschnittliche Dauer pro Befehl verlängert sich also um 60 %.

(6)

b) Der RISC-Prozessor verfügt über eine Prefetch Queue für bis zu 20 Befehle. Im Durchschnitt seien vier der vorab geholten Befehle Verzweigungsbefehle. Die Sprungzielvorhersage habe eine Trefferquote von 90%. Mit welcher Wahrscheinlichkeit befinden sich die benötigten Sprungbefehle in der Prefetch Queue?

Im Durchschnitt sind vier der 20 Befehle in der Prefetch Queue Sprungbefehle. Damit auch der letzte dieser Sprungbefehle der „richtige" ist, muß bei den drei vorigen Sprüngen die Sprungzielvorhersage das richtige Ziel ermittelt haben.

Die gesuchte Wahrscheinlichkeit liegt also bei $0{,}9^3 = 0{,}729$.

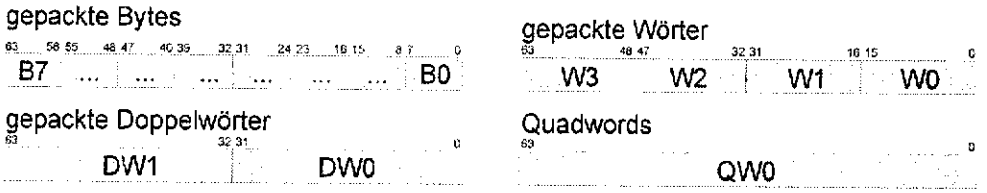
Hinweis:

Für den Fall, dass der aktuell in Ausführung befindliche Befehl ein Sprungbefehl ist, liegt die Wahrscheinlichkeit bei $0{,}9^4 = 0{,}6561$. Dieses Ergebnis wird nur dann als richtig akzeptiert, wenn die genannte Annahme in der Lösung genannt wurde.

(6)

Aufgabe 4 MMX-Rechenwerk (20 Punkte)

Ein MMX-Rechenwerk (*Multimedia Extension*) unterstützt gepackte Datenformate nach untenstehendem Bild, 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.

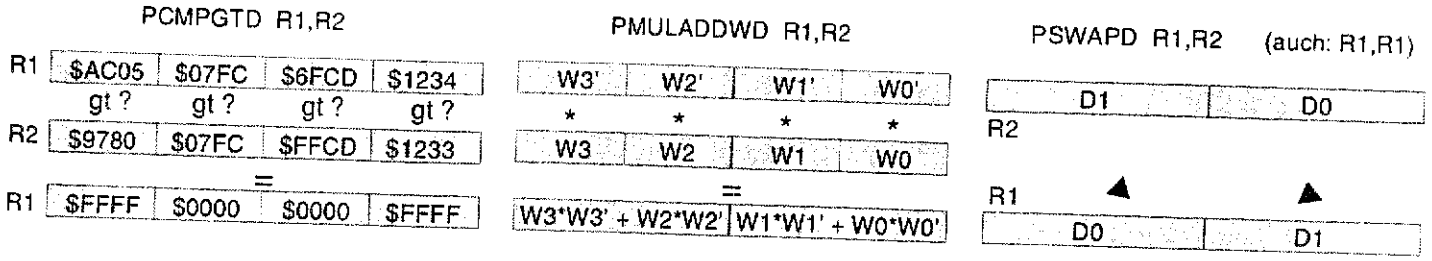


Der MMX Befehlssatz enthält u.a. die in folgender Tabelle angegebenen Befehle:

Mnemo		Bemerkung
	Logische Operationen (wirken auf alle 64 Bits)	
PAND	*Packed And*	bitweise Und-Verknüpfung
PANDN	*Packed And-Not*	bitweise Und-Verknüpfung mit negiertem 1. Operanden
POR	*Packed Or*	bitweise Oder-Verknüpfung
PXOR	*Packed Exclusive Or*	bitweise Antivalenz-Verknüpfung
	Vergleichsbefehle (für X = B, W, D, s. Skizze)	
PCMPEQX	*P. Compare Equal*	elementeweiser Vergleich auf gleich
PCMPGTX	*P. Compare Greater*	elementeweiser Vergleich auf größer
	Transferbefehle	
MOVQ	*Move Quadword*	Transfer zw. MMX-Reg. und MMX-Reg. bzw. Speicher
MOVD	*Move Doubleword*	Transfer zw. MMX-Reg. und MMX-Reg./Speicher
PSWAPD	*P. Swap Doubleword*	Vertauschen der Doppelwörter in Register, s. Skizze
	Multiplizier-Addierbefehl (W x W → D, s. nachfolgende Skizze)	
PMULADDWD	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).

Skizze:



1. In zwei 64-bit-Registern 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 paarweise (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).

   Bestimmung der Maximalwerte:

   a) MOVQ R3,R1 ; Kopie von R1 nach R3
   b) PCMPGTW R3,R2 ; Vergleich von R3 (R1) mit R2 auf größer, Ergebnis nach R3

   Hinweis: Aufgrund des Fehlers in der Skizze, wird auch PCMPGTD R3,R2 an dieser Stelle als richtige Lösung akzeptiert.

   c) PAND R1,R3 ; Maskieren von R1 mit R3, Ergebnis nach R1
   d) PANDN R3,R2 ; Invertieren der Maske in R3 und maskieren von R2, Erg. nach R3
   e) POR R1,R3 ; Oder-Verknüpfung von R1 und R3, Ergebnis nach R1

   a) R3 = ($C7A0, $50FE, $0800, $7F0B)
   b) R3 = ($FFFF, $0000, $0000, $FFFF)
   c) R1 = ($C7A0, $0000, $0000, $7F0B)
   d) R1 = ($0000, $FFFF, $ FFFF, $0000) ⇒ R1 = ($0000, $50FE, $A060, $0000)
   e) R1 = ($C7A0, $50FE, $A060, $7F0B)

   (8)

2. Geben Sie an, wie man mit dem Befehl PMULADDWD „doppelt-genaue" Multiplikationen 16 x 16 → 32 bit ausführen kann ?

   Es reicht, in einem der Operanden jedes zweite Wort auf $0000 zu setzen, z.B. W0 = W2 = $0000.
   Dann gilt für das Ergebnis: D1 = W3 * W3' + W2 * W2' = W3 * W3'
   D0 = W1 * W1' + W0 * W0' = W1 * W1'

   (4)

3. 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 Ausdruck

$$S = \sum_{i=0,..,7} Ci * Wi$$

berechnet werden.

Geben Sie eine MMX-Befehlsfolge für die geforderte Berechnung an.

Führen Sie die Befehlsfolge symbolisch mit den Bezeichnern Wi, Ci durch.

```
a) MOVQ       R1,Adr_1    ; Laden der ersten vier Wörter Wi
b) MOVQ       R2,Adr_1+4  ; Laden der zweiten vier Wörter Wi
c) MOVQ       R3,Adr_2    ; Laden der ersten vier Koeffizienten Ci
d) MOVQ       R4,Adr_2+4  ; Laden der zweiten vier Koeffizienten Ci
e) PMULADDWD  R1,R3       ; Multiplizieren der ersten vier Wörter/Koeffizienten mit
                              Addition
f) PMULADDWD  R2,R4       ; Multiplizieren der zweiten vier Wörter/Koeffizienten mit
                              Addition
g) PADDD      R1,R2       ; Addition der Teilergebnisse
h) MOVQ       R2,R1       ; Kopieren der Teilergebnisse nach R2
i) PSWAPD     R2,R2       ; Vertauschen der Teilergebnisse in R2
j) PADDD      R1,R2       ; Addition der Teilergebnisse im niederwertigen Doppel-
                              wort von R1
```

a) R1 = (W3, W2, W1, W0)
b) R2 = (W7, W6, W5, W4)
c) R3 = (C3, C2, C1, C0)
d) R4 = (C7, C6, C5, C4)
e) R1 = (C3*W3+C2*W2, C1*W1+C0*W0)
f) R2 = (C7*W7+C6*W6, C5*W5+C4*W4)
g) R1 = (C3*W3+C2*W2 + C7*W7+C6*W6, C1*W1+C0*W0 + C5*W5+C4*W4)
h) R2 = (C3*W3+C2*W2 + C7*W7+C6*W6, C1*W1+C0*W0 + C5*W5+C4*W4)
i) R2 = (C1*W1+C0*W0 + C5*W5+C4*W4, C3*W3+C2*W2 + C7*W7+C6*W6)
j) R1 = (C3*W3+C2*W2 + C7*W7+C6*W6 + C1*W1+C0*W0 + C5*W5+C4*W4,
C1*W1+C0*W0 + C5*W5+C4*W4 + C3*W3+C2*W2 + C7*W7+C6*W6)

Ergebnis:

k) C7*W7 + C6*W6 + C5*W5 + C4*W4 + C3*W3 + C2*W2 + C1*W1 + C0*W0

(8)

Aufgabe 5 Segmentierter Speicher (12 Punkte)

a) Erläutern Sie kurz die wesentlichen Merkmale der segmentorientierten Speicherverwaltung.

Bei der segmentorientierten Speicherverwaltung ist der virtuelle Adressraum in Segmente verschiedener Länge unterteilt. Jedem Prozeß sind ein oder mehrere Segmente, beispielsweise Code- und Daten-Segmente, zugeordnet. Jeder Auftrag kommt im Normalfall mit einer relativ kleinen Anzahl von Segmenten aus, deren Zuordnung durch den Benutzer oder durch den Compiler erfolgt. (3)

b) Im Kurs haben Sie die *First-fit*-Zuweisungsstrategie zur Einlagerung von Segmenten in den Arbeitsspeicher kennengelernt, bei der die freien Lücken im Speicherbereich nach aufsteigenden Anfangsadressen geordnet werden und das Segment in die erste Lücke eingelagert wird, in die es hinein paßt. In der Praxis wird häufig auch die alternative *Rotating-First-fit*-Strategie eingesetzt:

- Die Suche nach der ersten 'passenden' Lücke beginnt stets unmittelbar nach dem zuletzt eingelagerten Segment.
- Ist die Suche bis zum Ende des Speicherbereichs erfolglos, so wird sie am Beginn des Speicherbereichs fortgesetzt (*Round Robin*).
- Wird dabei der Ausgangspunkt der Suche wieder erreicht, so kann das Segment nicht eingelagert werden.

Das Betriebssystem kann jederzeit Speicherplatz freigeben, indem es die darin abgelegten Segmente auslagert.

Betrachtet werde nun ein 20 kbyte großer Speicherbereich, in dem die folgenden Segmente, deren Größe in kbyte angegeben ist, eingelagert werden sollen:

Segment	A	B	C	D	E	F	G	H	I	J
Größe	2	3	7	4	1	2	1	4	1	1

Es finde die nachstehende Folge von Ein- und Auslagerungen statt:

A B C <u>B</u> D E F <u>A</u> <u>C</u> G H I J <u>G</u> <u>F</u> B

Darin kennzeichne die Unterstreichung eines Segmentnamens die o.g. Auslagerung dieses Segments durch das Betriebssystem.

Tragen Sie in die Tabelle auf der folgenden Seite die Belegung des Speicherbereichs nach jeder Aktion ein, wobei Sie jedes Segment durch seinen Namen repräsentieren sollen. Dabei werde von einem zunächst völlig freien Speicherbereich

ausgegangen. Jedes Kästchen der Tabelle stehe für einen 1 kbyte großen Speicherausschnitt. Durch '-' werden freie Speicherplätze markiert.

Speicherbelegungsplan:

Aktion	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
A	A	A	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
B	A	A	B	B	B	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
C	A	A	B	B	B	C	C	C	C	C	C	C	-	-	-	-	-	-	-	-
B	A	A	-	-	-	C	C	C	C	C	C	C	-	-	-	-	-	-	-	-
D	A	A	-	-	-	C	C	C	C	C	C	C	D	D	D	D	-	-	-	-
E	A	A	-	-	-	C	C	C	C	C	C	C	D	D	D	D	E	-	-	-
F	A	A	-	-	-	C	C	C	C	C	C	C	D	D	D	D	E	F	F	-
A	-	-	-	-	-	C	C	C	C	C	C	C	D	D	D	D	E	F	F	-
C	-	-	-	-	-	-	-	-	-	-	-	-	D	D	D	D	E	F	F	-
G	-	-	-	-	-	-	-	-	-	-	-	-	D	D	D	D	E	F	F	G
H	H	H	H	H	-	-	-	-	-	-	-	-	D	D	D	D	E	F	F	G
I	H	H	H	H	I	-	-	-	-	-	-	-	D	D	D	D	E	F	F	G
J	H	H	H	H	I	J	-	-	-	-	-	-	D	D	D	D	E	F	F	G
G	H	H	H	H	I	J	-	-	-	-	-	-	D	D	D	D	E	F	F	-
F	H	H	H	H	I	J	-	-	-	-	-	-	D	D	D	D	E	-	-	-
B	H	H	H	H	I	J	B	B	B	-	-	-	D	D	D	D	E	-	-	-

(6)

c) Welchen Vorteil bietet das *Rotating-first-fit*-Verfahren gegenüber der *First-fit*-Strategie ?

Durch die zyklische Zuordnung der Segmente wird beim Rotating-first-fit-Verfahren die bei der First-fit-Strategie entstehende Konzentration kleiner Lücken am Anfang des Speicherbereichs vermieden.
Hierdurch müssen bei der Zuweisung größerer Segmente im Durchschnitt weniger Lücken auf ihre Größe hin untersucht werden, wodurch im Mittel eine Beschleunigung der Speicherzuweisung erreicht wird.

(3)

Aufgabe 6 Timer (20 Punkte)

Geben sei ein 16-bit-Zeitgeber/Zähler-Baustein (Timer) mit dem im folgenden Bild dargestellten Steuerregister CR. Die darin eingetragenen Bitfelder sollen im wesentlichen die im Kurs beschriebenen Funktionen haben.

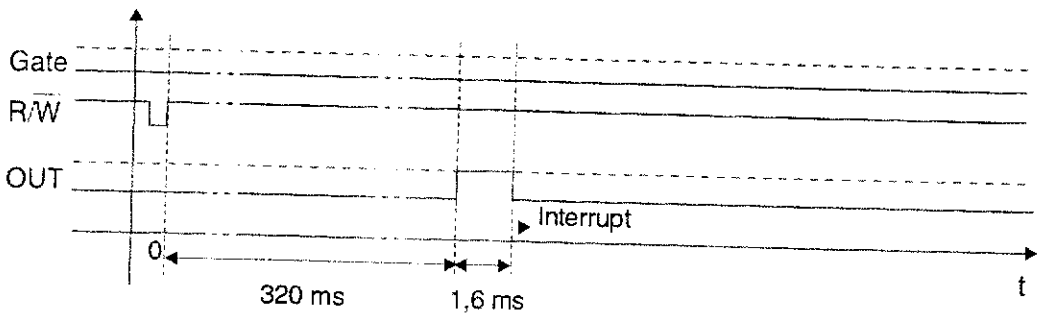
IE	OE	M2	M1	M0	CM	CL	PR

Es gelten die folgenden Abweichungen und Festlegungen:

- der Timer arbeitet nur mit einem einzigen Anfangswert, d.h. einem Auffangregister (*Latch*) pro Zähler; er ist re-triggerbar;
- das Rücksetz-Bit fehlt;
- durch IE=1 werden Interrupts zur CPU durchgelassen;
- durch OE=1 wird der Timer-Ausgang freigeschaltet;
- der Zähler kann nur dual im 16-bit-Modus (CM=1) oder 2x8-bit-Modus (CM=0) betrieben werden, nicht jedoch als Dezimalzähler;
- CL wählt zwischen einem externen 1 MHz-Takt (CL=1) oder dem internen Systemtakt (f >> 1MHz);
- durch PR=1 wird ein 1:8-Frequenzteiler aktiviert;
- durch die Bits M2, M1, M0 werden die folgenden Arbeitsweisen selektiert:

M2	M1	M0	Mode
0	0	0	periodischerZählmodus (Taktgenerator), nur Hardware-Triggerung
0	0	1	periodischerZählmodus, Hardware- und Software-Triggerung
0	1	0	Monoflop-Betrieb, nur Hardware-Triggerung
0	1	1	Monoflop-Betrieb, Hardware- und Software-Triggerung
1	0	0	*Strobe*-Betrieb, nur Hardware-Triggerung
1	0	1	*Strobe*-Betrieb, Hardware- und Software-Triggerung
1	1	0	Impulsmessung
1	1	1	Frequenzmessung

a) Am Ausgang OUT werde das folgende Zeitdiagramm (nicht maßstäblich) aufgezeichnet:



Geben Sie einen geeigneten Programmierwert für das o.g. Steuerregister CR sowie den Anfangswert für den Zähler an, die zu diesem Ausgangssignal führen können. (Der Wert für CR soll binär und hexadezimal, für das Latch dezimal und hexadezimal angegeben werden.) Begründen Sie Ihre Angaben.

Der Skizze ist zu entnehmen, daß das Signal durch das Lese-/Schreibsignal R/W getriggert und nicht periodisch ausgegeben wird. Nach den Ausführung im Kurs kann daher nur der Monoflop-Betrieb im 2x8-Zählmodus vorliegen.
Die Länge des positiven Impulses IL wird durch das L-Byte des Zählers gegeben. IL kann maximal 256 Zählschwingungen lang sein, bei einem externem 1-MHz-Takt[1] wären das nur 256 µs. Also muß der interne Frequenzteiler 1:8 aktiviert sein. Aus IL = 1,6 ms = 8*200 µs ergibt sich damit ein Wert für den LSB-Zähler von AW_{LSB} = 200-1 = 199 = $C7. Aus der Zeit V = 320 ms bis zur Ausgabe des Impulses folgt dann:

$$V = (AW_{MSB} + 1) * IL \Rightarrow AW_{MSB} = V / IL - 1 = 200 - 1 = 199 = \$C7.$$

Also muß das Auffangregister vor dem Zähler mit dem Wert $C7C7 geladen werden.
Für das Steuerregister ergibt sich:

IE	OE	M2	M1	M0	CM	CL	PR
1	1	0	1	1	0	1	1

d.h. CR = 1101 1011 = $ DB.

(3)

b) Geben Sie die Werte für CR und Latch für den Fall an, daß ein symmetrisches Rechtecksignal (*Square Wave*) mit minimaler Frequenz am Ausgang OUT erzeugt wird. Wie groß ist diese Frequenz ? Wie groß seine Schwingungsdauer ? Setzen Sie dabei voraus, daß der Ausgang OUT (erst) nach jedem vollständigen Zählzyklus des Timers seinen Zustand wechselt.

Der minimalen Frequenz entspricht die maximale Schwingungsdauer, die beim maximalen Anfangswert des Zählers $FFFF erreicht wird. Im 16-bit-Modus werden 65536 Takte bis zum Zustandswechsel am Ausgang OUT gezählt. Im 2x8-bit-Modus findet bereits während des Zählzyklus der Übergang statt. Durch den 1:8-Teiler wird die Länge des Zählzyklus vergrößert. Nach Voraussetzung ist die Schwingungsdauer des externen Taktes größer als des Systemtaktes. Aus diesen Angaben ergibt sich für die Programmierung des Steuerregisters CR:

IE	OE	M2	M1	M0	CM	CL	PR
0/1	1	0	0	1	1	1	1

[1] Nach Voraussetzung wäre die Zeit für den höher frequenten internen Takt noch kürzer.

d.h. CR = $4F oder CR = $CF.

Für die max. Schwingungsdauer gilt: D = 2 * (AW+1) * 8 * 1 µs = 1.048.576 µs ≈ 1s.
Die minimale Frequenz ergibt sich daraus näherungsweise zu 0,95 Hz.

(5)

c) Der Prozessor hat keine Hardware-Möglichkeit festzustellen, ob der von ihm im periodischen Zählmodus gestartete Timer über seinen Gate-Eingang angehalten wurde oder nicht. Geben Sie die Skizze für ein Programm(stück) an, durch das der Prozessor den Zustand des *Gates* (zu einem bestimmten Zeitpunkt) feststellen kann.

Der Zustand des „laufenden" Zählers im Timer muß sich spätestens nach der maximalen Taktdauer von 8 * 1 µs ändern. Im Programm(stück) kann daher der Zählerzustand gelesen und in einem Register abgelegt werden. Das Programm muß danach eine Zeitverzögerung von wenigstens 8 µs erzeugen (z.B. durch eine Zählschleife oder einen weiteren Timer) und dann den neuen Zählerzustand ermitteln. Nur dann, wenn dieser mit dem abgespeicherten Wert übereinstimmt, war der Zähler zwischenzeitlich gestoppt.

(4)

d) Geben Sie die Skizze für ein Programm(stück) an, das den Timer als sog. Wobbel-Generator betreibt, d.h. die Frequenz der Rechteckschwingung am Ausgang OUT steigt von einer (programmierbaren) Startfrequenz bis zu einer Endfrequenz an und fällt danach wieder zur Anfangsfrequenz ab. Dieser Vorgang wird zyklisch fortgesetzt, wobei die Zeitspanne für einen Zyklus variabel ist.

Skizze:

1. Die Anfangswerte für Start- und Endfrequenz werden in Pufferregister abgelegt,
2. der Timer wird mit dem Wert CR = $CF programmiert,
3. der Startwert wird in das Latch geschrieben und dadurch der Timer initialisiert,
4. nach einer bestimmten Anzahl von Aufrufen der Interruptroutine zum Timer wird der Wert im Latch um 1 erhöht; diese Anzahl muß gerade sein, um jeweils Vollschwingungen des Ausgangssignals OUT zu berücksichtigen,
5. nach Erreichen des Wertes der Endfrequenz wird der Latch-Wert – analog zu 4. – erniedrigt, bis wiederum der Startwert erreicht wird.

(5)

e) Programmieren Sie den Timer als Watch-Dog, der nach 1 ms einen Alarm (Interrupt) auslöst, wenn er nicht rechtzeitig wieder getriggert wird.

Hier muß der Strobe-Modus mit Software-Triggerung und aktiviertem Interrupt gewählt werden. Der Ausgang muß nicht aktiviert sein, da der Timer den Alarm über seinen Interrupt-Ausgang zur CPU melden kann. Also ergibt sich:
CR = 1010 1110, Dauer: 1 ms = (AW+1) * 1 µs ⇒ AW = 999 = $03E7.

(3)