Minimalsystem
CPU, Adresseneingabe, Dateneingabe,
Speicher (Schalter auf 1), Port

HEX-Tastatur nicht angeschlossen!

BEDIENUNGSANLEITUNG: MINIMALSYSTEM
==========

Vor Anlegen der Betriebsspannung:

CPU-PLATINE	- S1 auf OSZILLATOR
	- S2 auf 1 Hz
	- S3 auf SCHRITT
	- S4 auf RUN
DATENEINGABE	- S auf DAFLOT

* Das Gerät einschalten!

Normalerweise leuchten jetzt auf der CPU-Platine OPACK, M/IO, RUN/WAIT und WRP. Die FLAG-LED kann zufällig leuchten. CLOCK blinkt im 1 Hz-Takt und die LED OPREQ leuchtet nach spätestens 6 Clockpulsen.
Auf der Dateneingabe leuchten alle roten LEDs über den Datenschaltern - unabhänging von der zufälligen Stellung dieser Schiebeschalter. Es leuchten die grünen LEDs ADPER und WRITE. ADMEM bleibt dunkel.
Es kann sein, daß auf der CPU die LED RUN/WAIT ausnahmsweise auf Nullsignal liegt (nicht leuchtet) - was der Fall ist, wenn der Prozessor in den WAIT-Zustand gelaufen ist. Aus diesem kann er nur mit RESET erlöst werden. Die RESET-Taste drücken und mindestens 3 Clockpulse festhalten!
Durch RESET wird der interne Programmzähler im Mikroprozessor auf Null gesetzt. Es werden keine Registerinhalte oder gar Programme, wie bei BASIC-Rechnern üblich, durch ein RESET gelöscht.

Eingeben und Befehlsverarbeitung

* Schiebeschalter auf der Dateneingabe auf WRITE!
  Die grüne LED WRITE erlischt (aktiv 0), ebenso alle roten LEDs über den Datenschaltern.

* Datenschalter gemäß erstem Befehl setzen!
  Die LEDs auf der Eingabe ermöglichen eine optische Kontrolle, dabei signalisiert eine leuchtende LED eine logische 1.

* Einzelschrittaste STEP drücken!
  Auf der CPU-Platine gehen zunächst OPACK, dann OPREQ auf Null. Die LEDs erlöschen. Wenn beide LEDs wieder 1 sind, wartet der Prozessor auf den nächsten Befehl.

OPACK = OPERATION ACKNOWLEDGE (aktiv 0) - "Ich quittiere den vorangegangenen Befehl!"

OPREQ = OPERATION REQUEST (aktiv 1) - "Ich bin mit der Verarbeitung des letzten Befehls fertig, gib den nächsten ein!"

* Datenschalter neu setzen. Einzelschrittaste drücken - usw.

Auslesen

* Wenn ein Ausgabebefehl erteilt wird, z.B. um das Ergebnis einer Addition anzuzeigen, dann gehen die I/O-LED auf der CPU und die grüne LED ADPER (beide aktiv 0) auf 0 und zeigen an, daß der Prozessor den Befehl verstanden hat und einen PORT ansteuert.

* Jetzt wird der Schalter S auf der Dateneingabe von WRITE auf DAFLOT gelegt. Dadurch werden die 8 LEDs auf den Datenbus geschaltet und stehen dem Prozessor zur Anzeige zur Verfügung. Die Stellung der Datenschalter hat keinen Einfluß auf die Ausgabe.

Programm fortsetzen

* Achtung! Nach einem Ausgabebefehl benötigt der Prozessor noch einen Arbeitstakt, um mit der Befehlsverarbeitung zu Ende zu kommen. In der Stellung DAFLOT betätigen wir noch einmal die STEP-Taste. I/O und ADPER werden wieder 1, ADMEM wird 0. Das Aufleuchten aller 8 LEDs auf der Dateneingabe zeigt uns an, daß der Datenbus wieder frei ist.

* Den Schalter S wieder auf WRITE stellen und wie bei Einlesen und Befehlsverarbeitung fortfahren.

Bei der Verarbeitung der einzelnen Befehle werden immer mehrere Steuersignale von der CPU ausgelöst. Unser System macht die wichtigsten sichtbar. Beobachtbar sind die Signale aber nur, wenn die CPU im 1 Hz-Takt betrieben wird. Die Steuersignale ADREN, DBUSEN und PAUSE werden nicht optisch angezeigt. Über die Bedeutung dieser Signale gibt das Begleitbuch im Kapitel 7 Auskunft.

****
Wenn man diese Bedienungsschritte an einem kleinen Programm übt, wird man bald sehr viel Freude an dieser "Minimalversion" eines Computers haben!
****

EINSCHREIBEN UND AUSLESEN VON DATEN

Bevor der Mikroprozessor, d.h. dessen ALU, Daten verknüpfen kann, müssen ihm diese auf dem Datenbus angeboten werden. Bietet man die Daten (also irgendeine 8-bit-Zahl) direkt nach dem Ladebefehl an, so nennt man das **unmittelbares Laden** (immediate load). Der 2650 gestattet das unmittelbare Laden aller Arbeitsregister (R0 - R3) mit dem 2-Byte-Befehl **LODI,r** v. LODI ist der **Assemblerbefehl** für unmittelbares Laden. Das kleine r nach dem Komma ist der Platzhalter für die Registerbezeichnungen, für die wir uns auf R0,R1,R2 und R3 einigen. Das kleine v steht für Wert (engl. value). Im Assembler geben wir den Binärwert in seiner hexadezimalen Schreibweise an. Weil wir gewohnt sind, Zahlen dezimal zu schreiben und zu lesen, kann es anfangs zu Schwierigkeiten bei der Codierung der Befehle kommen. Wenn man beim Notieren des Wertes v ein großes H vor die Zahl setzt, z.B. H10, ist der hexadezimale Wert 10 (Eins-Null), nicht der dezimale Wert 10 (Zehn) gemeint. Die Dezimalzahl 10 entspricht dem Binärwert 0A.
LODI,R0 H0A steht in der Assemblerschreibweise des 2650 für die Anweisung: "Lade Register 0 unmittelbar mit dem Wert 0A - der Dezimalzahl 10 !"
Damit unser Minimalsystem diesen Befehl verstehen kann, müssen wir den dualen Befehlscode - den Opcode - mit Hilfe unserer Befehlsliste ermitteln. Auf Seite 314 im Begleitbuch finden wir den **LOAD IMMEDIATE** - Befehl beschrieben. Das erste Byte enthält den Ladebefehl selbst. Die Bitpositionen 0 und 1 sind für die **binäre Registerkennung** freigelassen. Hier setzen wir für R0 00 ein. Die Kennungen für die anderen Register sind 01 für R1, 10 für R2 und 11 für R3. Für LODI,R0 ermitteln wir also den Dualcode 00000100 . Gemäß der Bedienungsanleitung für das Minimalsystem stellen wir den ersten Befehl an den 8 Schiebeschaltern auf der Dateneingabe ein. Der dritte Schalter von rechts wird in die EIN-Stellung gebracht (nach oben schieben!), alle anderen Datenschalter verbleiben in der AUS-Stellung. Die dritte LED leuchtet und ermöglicht somit eine optische Kontrolle unserer Eingabe. Wir betätigen kurz die Einzelschrittaste **STEP**! Wenn die CPU mit der Befehlsverarbeitung fertig ist, signalisiert sie dies, in dem sie die Steuerleitungen OPACK und OPREQ auf logisch 1 legt. Die beiden LED neben CLOCK leuchten. Der Prozessor "wartet" auf den nächsten Befehl, nämlich die Daten für Register R0! Wir stellen an den Datenschaltern 00001010 ein - das ist der Dualcode für den Dezimalwert 10 - und betätigen die STEP-Taste ein zweites Mal. Der Prozessor akzeptiert den Befehl und lädt die Daten nach R0. Der ursprüngliche Inhalt von R0 wird überschrieben. Vertrauen ist gut, Kontrolle ist besser; das gilt auch für den Mikroprozessor. Deshalb wollen wir den Prozessor veranlassen, den Inhalt von Register R0 wieder auf den Datenbus auszugeben, der bei unserem Minimalsystem über die 8 LEDs angezeigt wird. Wir benötigen einen **AUSGABE-Befehl**.

Die AUSGABE-Befehle dienen dem Datentransport zwischen den CPU-Registern und den Ausgabekanälen, die in der Fachsprache auch als PORTs bezeichnet werden. Man spricht von Output- und Input-Ports, wenn man spezielle Bausteine für die Eingabe bzw. Ausgabe von Daten zur Verfügung hat. Das Minimalsystem arbeitet ohne besondere Portbausteine. Wir haben aber trotzdem die Möglichkeit, die Wirkung der AUSGABE-Befehle zu zeigen. Wir "tun" so, als wären die beiden 8-Bit-Ports **DATA** und **CONTROL** an den Datenbus angeschlossen. Die Bezeichnung der Ports stammt vom Hersteller des 2650 und ist eigentlich bedeutungslos. Jedesmal wenn diese Ports durch entsprechende Befehle angesprochen werden, legt der Prozessor die für die Ports bestimmten Daten auf den Datenbus. Im Einzelschrittmodus brauchen wir dann lediglich den Schalter DAFLOT/WRITE auf der Dateneingabe in die Stellung DAFLOT zu bringen, um in Ruhe die Daten betrachten zu können.

Der AUSGABE-Befehl für den Port **DATA** ist im Begleitbuch auf Seite 348 beschrieben. Die Assemblerschreibweise ist **WRTD,r** - dies steht für "WRITE DATA", was soviel heißt wie "Schreibe in den Port DATA!" Das kleine r steht wieder für die Register, also muß der vollständige Assemblerbefehl für die Ausgabe des Inhalts von Register R0 WRTD,R0 lauten. Im Dualcode des Befehls sind wieder die Bitpositionen 0 und 1 noch nicht belegt. Wir tragen hier die binäre Kennung für R0 ein, also 00, und erhalten den Opcode 11110000. Wenn wir alternativ den Befehl **WRTC,R0** (Seite 348 unten) verwenden, ergibt sich der Opcode 10110000.

Wir stellen einen der beiden Befehle an den Datenschaltern ein und betätigen STEP. Die I/O-LED auf der CPU-Platte und die grüne LED ADPER auf der Dateneingabe gehen auf 0 und zeigen an, daß der Prozessor den Befehl verstanden hat und einen Port ansteuert. Auf der Dateneingabe erlöschen meistens alle 8 LEDs. Allerdings können auch einzelne LED leuchten. Dieser Fall tritt ein, wenn durch den AUSGABE-Befehl eine Datenleitung auf 1 gelegt wird und gleichzeitig der Datenschalter dieser Leitung eingeschaltet ist. Wir stellen den Schiebeschalter auf **DAFLOT**. Der Wert 00001010 erscheint auf der Anzeige. Unser Mikroprozessor gehorcht also aufs Wort.

Wollen wir die Programmierung fortsetzen, um z.B. die übrigen Register R1,R2 und R3 zu laden, müssen wir den Rechner erst aus dem Ausgabemodus zurückholen. In der Stellung DAFLOT betätigen wir noch einmal kurz die STEP-Taste. Die LEDs I/O und ADPER werden wieder 1, ADMEM wird 0, gleichzeitig leuchten alle 8 LEDs auf der Dateneingabe auf, damit wird die Freigabe des Datenbusses angezeigt. Wir schalten erst jetzt den Schalter DAFLOT/WRITE auf WRITE und können mit dem Einschreiben weiterer Befehle fortfahren.

Eine Zusammenstellung der LODI-, der WRTD und WRTC-Befehle ist mit Hilfe der Befehlsliste im Begleitbuch leicht möglich:

LODI,R0 0000 0100

LODI,R1 0000 0101

LODI,R2 0000 0110

LODI,R3 0000 0111

Die Lücke zwischen je 4 Bit beim Dualcode erleichtert die Umwandlung in den entsprechenden hexadezimalen Code. Außerdem verbessert sich dadurch die Lesbarkeit der achtstelligen Zahl. Den Hexcode der LODI-Befehle findet man in der tabellarischen Befehlsliste im Anhang des Begleitbuches (Seite 355 - 356). In der zweiten Zeile der Tabelle sind die LODI-Befehle in der Gruppe der LOAD/STORE - Befehle aufgelistet:

LODI,R0 04 LODI,R1 05 LODI,R2 06 LODI,R3 07

Die beiden Ein-Byte-Ausgabebefehle für alle Register sind:

WRTD,R0 1111 0000	WRTC,R0 1011 0000
WRTD,R1 1111 0001	WRTC,R1 1011 0001
WRTD,R2 1111 0010	WRTC,R2 1011 0010
WRTD,R3 1111 0011	WRTC,R3 1011 0011

Den Hexcode können wir auf Seite 356 nachschlagen:

WRTD,R0 F0 WRTD,R1 F1 WRTD,R2 F2 WRTD,R3 F3

WRTC,R0 B0 WRTC,R1 B1 WRTC,R2 B2 WRTC,R3 B3

Wir haben jetzt schon drei Befehle aus dem 75 Befehle umfassenden Befehlsvorrat des 2650 kennengelernt. Um sie noch einmal zu üben, geben wir ein kleines Programm ein, das außerdem eine gute Übung der Bedienung des Minimalsystems darstellt:

Befehl 1:	LODI,R0 H05	0000 0100
		0000 0101
Befehl 2:	LODI,R1 H10	0000 0101
		0001 0000
Befehl 3:	LODI,R2 H1A	0000 0110
		0001 1010
Befehl 4:	LODI,R3 HFE	0000 0111
		1111 1110
Befehl 5:	WRTD,R0	1111 0000
Befehl 6:	WRTD,R1	1111 0001
Befehl 7:	WRTD,R2	1111 0010
Befehl 8:	WRTD,R3	1111 0011

Hat das Übungsprogramm richtig funktioniert? Erfahrungsgemäß machen die Bedienungsschritte beim Auslesen der Daten mit dem WRTD oder WRTC-Befehl die meisten Schwierigkeiten. Man vergißt leicht, den Schalter DAFLOT/WRITE nach dem Ausgabebefehl in die Stellung DAFLOT zu bringen. Die nächste Fehlerquelle liegt in der Tatsache, daß der Prozessor nach der Ausgabe noch einen zusätzlichen STEP benötigt, um den Befehl abschließen zu können. Wir müssen also unbedingt in der Stellung DAFLOT noch einmal die Einzelschrittaste betätigen. Erst dann darf der Schalter von DAFLOT (AUSGABE) wieder auf WRITE (EINGABE) umgestellt werden und der nächste Befehl an den Datenschaltern eingestellt werden!

* Bitte noch einmal die Anweisungen der Bedienungsanleitung zum
* Minimalsystem zum "Auslesen" und "Programm fortsetzen" lesen!

Im Minimalsystem lassen sich noch zwei weitere DATENTRANSPORT-BEFEHLE verwenden. Es handelt sich um den **Ladebefehl** LODZ,r (Load register zero) und um den **Speicherbefehl** STRZ,r (Store register zero).
Beim Befehl LODZ,r , z.B. LODZ,R1, wird Register R0 mit Daten aus Register R1 geladen. R1 ist die **Quelle** der Daten, R0 das **Ziel**. Der ursprüngliche Inhalt von R0 wird überschrieben, der Inhalt des Quellregisters bleibt unverändert.
Den Binärcode für den **LOAD REGISTER ZERO** Befehl finden wir in der Befehlsliste im Begleitbuch auf Seite 314. Weil R1, R2 und R3 als Quellregister eingesetzt werden können, ergeben sich folgende Dualcodes:

LODZ,R1 0000 0001

LODZ,R2 0000 0010

LODZ,R3 0000 0011

Die Hexcodes sind 01, 02 und 03.

Wenn man einen Wert aus Register R0 in ein anderes Arbeitsregister der CPU transportieren möchte, benutzt man den **STORE REGISTER ZERO** Befehl STRZ,r. Mit STRZ,R2 kann man den Inhalt von R0 in das Register R2 kopieren. Diesmal ist also R0 **Datenquelle** und R2 **Ziel**. Der Inhalt von R0 überschreibt den alten Inhalt von R2. R0 wird nicht verändert. Auf Seite 316 des Begleitbuches finden wir die Befehlsbeschreibung und den Binärcode:

STRZ,R1 1100 0001

STRZ,R2 1100 0010

STRZ,R3 1100 0011

Die Hexcodes sind demnach C1, C2 und C3.

Mit einem kleinen Übungsprogramm sollte man die beiden neuen Befehle ausprobieren:

Befehl 1:	LODI,R0 H07	0000 0100
		0000 0111
Befehl 2:	STRZ,R1	1100 0001
Befehl 3:	WRTD,R1	1111 0001

* Jetzt muß der Inhalt von R1 richtig ausgelesen werden! Wenn
* der Schalter S der Dateneingabe von WRITE auf DAFLOT gelegt
* wird, erscheinen die Daten 0000 0111 an den 8 LEDs. Wir
* setzen das Programm fort, in dem wir in der Stellung DAFLOT
* noch einen STEP geben und dann auf WRITE umschalten. Der
* nächste Befehl kann eingestellt werden.

Befehl 4:	LODI,R2 H22	0000 0110
		0010 0010
Befehl 5:	LODZ,R2	0000 0010
Befehl 6:	WRTC,R0	1011 0000

* Der Inhalt von R0 wird diesmal mit dem WRTC-Ausgabebefehl
* ausgelesen. Es erscheint der Binärwert 0010 0010 auf dem
* Datenbus.

Im nächsten Kapitel wollen wir uns den **DATENVERARBEITUNGSBEFEHLEN** zuwenden. Zu den wichtigsten Befehlen aus dieser Gruppe gehören die arithmetischen Befehle zur **ADDITION** und **SUBTRAKTION**. Mit ihnen läßt sich zeigen, daß unser Gerät die Bezeichnung COMPUTER (engl. to compute = errechnen, berechnen, auswerten) verdient.

****
****

EINFACHE RECHENPROGRAMME MIT DEM MINIMALSYSTEM

Wir wollen uns zunächst auf Additionen beschränken, bei denen die Summe nicht größer werden kann als

	1111 1111,
also	FF (hexadezimal),
also	255 (dezimal).

Dann tritt beim höchstwertigen Bit (most significant bit) kein Übertrag auf.
Versuchen wir es einmal mit der einfachen Additionsaufgabe

5 + 7 = ?

Wir laden ein Arbeitsregister der CPU **unmittelbar** mit dem Binärwert des ersten Summanden, z.B. R0 durch den Befehl **LODI,R0 H05**. Zum Inhalt von R0 können wir **unmittelbar** den zweiten Summanden addieren. Wir benötigen dafür den Befehl ADDI,R0 H07. Die Abkürzung **ADDI** steht für **ADD IMMEDIATE** und bezeichnet einen 2-byte-Befehl, mit dem ein Binärwert zum Inhalt des im Befehl genannten Registers addiert werden kann. Der Dualcode für diesen Befehl kann der Befehlsliste im Begleitbuch (S. 318) entnommen werden:

ADDI,R0	1000 0100	hexadezimal 84
ADDI,R1	1000 0101	hexadezimal 85
ADDI,R2	1000 0110	hexadezimal 86
ADDI,R3	1000 0111	hexadezimal 87

Zum vollständigen ADDI-Befehl gehört, ähnlich den LODI-Befehlen, immer das zweite Befehlsbyte mit dem Binärwert, der addiert werden soll.
Jetzt können wir das kleine Rechenprogramm schreiben und einlesen.

Befehl 1:	LODI,R0 H05	0000 0100	hex 04
		0000 0101	hex 05
Befehl 2:	ADDI,R0 H07	1000 0100	hex 84
		0000 0111	hex 07
Befehl 3:	WRTD,R0	1111 0000	hex F0

Der Mikroprozessor befolgt "brav" unsere Befehle und gibt nach Befehl 3 das Resultat 12 in binärer Form (0000 1100) aus.
Aber, beim Erstkontakt mit Mikroprozessoren ist man vor Überraschungen nicht sicher! Sollte Ihr Rechner der Meinung sein, daß 13 (0000 1101) das richtige Ergebnis ist, müssen sie das Programm noch einmal eingeben. Beim zweiten Versuch erhalten Sie garantiert das "richtige" Ergebnis , vorausgesetzt, Sie haben den Rechner richtig bedient! Die Erklärung für dieses seltsame Verhalten folgt auf der nächsten Seite.

Jeder ordentliche Mikroprozessor hat auf seiner Zentraleinheit ein sogenanntes STATUSREGISTER. Dieses besondere Register dient dem Rechenwerk zum Notieren des Zustandes (lat. status) des Rechners nach bestimmten Operationen. Es kann dabei den nächsten Befehl beeinflussen. Eine der möglichen Notizen, die der Rechner in seinem Statusregister ablegt, ist z.B. das Speichern eines Übertrags bei einer Addition. Jedesmal wenn eine Addition ein größeres Ergebnis liefert als hexadezimal FF, wird im Statusregister das sogenannte CARRY-Bit beeinflußt. Der Programmierer kann aber auch Einfluß auf das Statusregister nehmen, und z.B. dem Rechner vorschreiben, einen eventuell gespeicherten Übertrag zu berücksichtigen oder zu ignorieren.

Beim 2650 existieren zwei 8-Bit Statusregister. Sie werden als OBERES bzw UNTERES PROGRAMM-STATUS-WORT bezeichnet. Die englischen Bezeichnungen lauten Program Status Upper (PSU) und Programm Status Lower (PSL). Uns interessiert zunächst nur das PSL! Wer sich schon vorinformieren möchte, sollte die Seiten 114 - 117 im Begleitbuch durcharbeiten.

Für das seltsame Verhalten des Rechners bei Additionen sind die Bits 0 und 3 im UNTEREN PROGRAMM-STATUS-WORT (PSL).

* Bit 0 ist das CARRY-Bit es speichert beim Abarbeiten von
* Additions-, Subtraktions- oder Rotationsbefehlen den Übertrag
* aus dem Bit 7 des benutzten Arbeitsregisters.

* Bit 3 ist das WITH/WITHOUT CARRY-Bit (WC). Es steuert die oben
* genannten Befehle. Ist WC = 0, so wird bei der Durchführung
* eines Additions-, Subtraktions- oder Rotationsbefehls ein
* vorhandenes Carry nicht berücksichtigt. Ist WC = 1, so wird
* z.B. bei einer Addition das Carry-Bit automatisch zum Ergebnis
* addiert.
* Unabhängig vom WC-Bit wird aber immer das Carry-Bit (Bit 0)
* automatisch berechnet.

Beim Einschalten des Rechners nehmen die Register auf der CPU einen zufälligen Inhalt an - sie sind also nicht leer oder gelöscht! Es ist also denkbar, daß Bit 0 und Bit 3 im PSL beim Einschalten der Stromversorgung gesetzt wurden (beide 1). Für unsere einfache Rechenaufgabe hatte dies fatale Folgen. Der Rechner berücksichtigte den zufälligen Übertrag und erhielt ein Ergebnis, das um 1 zu groß war. Er berechnete den Übertrag neu - das Resultat 0000 1101 erzeugte keinen Übertrag, weil es kleiner als 1111 1111 ist - und setzte das CARRY auf 0! Bei der Wiederholung des Programms mußte logischerweise das richtige Ergebnis berechnet werden, der zu berücksichtigende Übertrag war ja inzwischen 0 geworden!

* Wie kann man das PSL durch Programmbefehle beeinflussen, um
* bei Additionen im Zahlenraum bis hexadezimal FF ohne Übertrag
* zu rechnen?

BEEINFLUSSUNG DES UNTEREN PROGRAMM-STATUS-WORTES

Die Programmierung des Programm-Status-Wortes ist für den Programmieranfänger wegen der vielen Möglichkeiten nicht ganz einfach. Die einzelnen Bitpositionen können durch Befehle gelöscht, gesetzt und sogar auf das Vorhandensein von Null- bzw. Einssignal getestet werden.
Für die ersten Übungsprogramme mit dem MINIMALSYSTEM genügt es, wenn wir lediglich das CARRY und das WITH CARRY-Bit beeinflussen.

Beispiel 1:
Bei Additionen und Subtraktionen soll der Zahlenraum eines 8-Bit Registers nicht überschritten werden. Das größte zugelassene Ergebnis ist hexadezimal FF. Ein zufällig vorhandener Übertrag soll gelöscht werden.
Um die Beispielbedingungen zu erfüllen, müssen im PSL Bit 0 und Bit 3 gelöscht werden. Alle anderen Bits im PSL haben keinen Einfluß und werden ebenfalls auf Null gesetzt.
Zur Änderung des PSL laden wir zunächst Register R0 mit dem LODI-Befehl mit dem gewünschten Inhalt des PSL, also H00. Dann wird mit dem LPSL-Befehl der Inhalt von R0 in das PSL kopiert.
LPSL steht für Load Program Status Lower.
Der LPSL-Befehl wirkt nur auf Register R0, der Inhalt der anderen Register kann nicht direkt in das PSL kopiert werden. Die Befehlsbeschreibung finden wir auf S.331 im Begleitbuch:

```
    LPSL     1001 0011     hexadezimal 93
```

Die vollständigen Programmschritte wären also:

```
    LODI,R0 H00     0000 0100     hex 04
                    0000 0000     hex 00
    LPSL            1001 0011     hex 93
```

Beispiel 2:
Der Rechner soll für eine 16-Bit Addition programmiert werden. Dafür ist die Berücksichtigung des Übertrags zu programmieren. Das CARRY muß vor der ersten Addition gelöscht sein. Wir setzen Bit 3 WC auf 1 - With Carry "Berücksichtige einen Übertrag!" - und Bit 0 C auf 0 - C Carry "Lösche einen Übertrag!"

Die Programmschritte sind :

```
    LODI,R0 H08     0000 0100     hex 04
                    0000 1000     hex 08
    LPSL            1001 0011     hex 93
```

Etwas ungewöhnlich ist die Berücksichtigung eines Übertrags bei Subtraktionen. Meistens bezeichnet man diesen Übertrag als BORROW. Der 2650 setzt das CARRY (in diesem Fall BORROW) dann auf 1, wenn kein Übertrag von der nächsthöheren Stelle, also kein Borgen, erforderlich war. Diese Tatsache berücksichtigen wir im nächsten Beispiel.

Beispiel 3:
Der Rechner soll für eine 16-Bit Subtraktion programmiert werden. Das WC-Bit und das C-Bit müssen am Programmanfang auf Eins gesetzt werden. Merke : C-Bit = 1 bei Subtraktion bedeutet, daß kein Borgen erforderlich war!

Die Programmschritte sind:

```
    LODI,R0 H09     0000 0100     hex 04
                    0000 1001     hex 09
    LPSL            1001 0011     hex 93
```

* Wenn die im Carry-Bit gespeicherte Information mit in den folgenden Rechenvorgang übernommen wird, kann man mit beliebig großen Zahlen rechnen, die sich dann über mehrere Bytes erstrecken können. Ein solches Programm wird im folgenden Kapitel gezeigt werden.

Der 2650 gestattet es auch, daß man den Inhalt des PSW ausgibt. Die Betrachtung der einzelnen Bitpositionen kann dabei beim Verstehen der Arbeitsweise des Prozessors helfen.
Mit dem **SPSL-Befehl (Store Program Status Lower)** kann man das PSL nach Register R0 kopieren. Mit einem WRTD oder WRTC-Befehl ließe sich dann das PSL über R0 ausgeben. Die Befehlsbeschreibung und den Binärcode finden wir auf S.332 im Begleitbuch.

```
    SPSL     1000 0011     hexadezimal 13
```

Die Befehlsfolge :

```
    SPSL          1000 0011     hex 13
    WRTD,R0       1111 0000     hex F0
```

gestattet also die Betrachtung des Inhalts des PSL.

RECHNEN MIT GROSSEN ZAHLEN : 16-Bit ADDITION UND SUBTRAKTION

Wenn man die normale Binärzahlengröße um 8 auf 16 Stellen erweitert, können dezimale Werte von bis zu 65535 binär dargestellt werden. Die Rechenoperationen Addition und Subtraktion ändern sich durch die größere Zahlenbreite nicht.

Eine 16 Bit Zahl besetzt zwei 8 Bit Speicherplätze. Im Minimalcomputer müssen wir deshalb die Zahlen in zwei Hälften auf zwei der vier Arbeitsregister verteilen. Die zwei Hälften nennen wir **LOW-BYTE** und **HIGH-BYTE**. Das LOW-BYTE enthält den niederstwertigen Teil der Zahl, das HIGH-BYTE den höherwertigen. 16 Bit Rechnungen bezeichnet man in der Computersprache oft als Rechnungen in **doppelter Genauigkeit** (DOUBLE PRECISION), dieser Ausdruck ist leider etwas mißverständlich.

Bevor wir unseren Rechner programmieren, zunächst ein theoretisches Beispiel für eine 16 Bit Addition:

Summand A	0000 0001 1000 0001	= 385 dezimal
Summand B	+ 0000 0001 1000 0101	= 389 dezimal
Übertrag	+ 11 1	11
Summe	0000 0011 0000 0110	= 774 dezimal

Von besonderer Bedeutung ist der Übertrag, der bei unserem Beispiel bei der Addition der Bit 7 von A und B entsteht. Er muß bei der folgenden Addition berücksichtigt werden. Beim Rechnen mit Bleistift und Papier ist das kein Problem, unser Rechner dagegen muß in der Lage sein, diesen Übertrag zu speichern und zu verarbeiten. Das Addierwerk der CPU speichert einen auftretenden Übertrag bei Bit 7 im Bit 0 des PSL. Dieses Bit - CARRY genannt - wird immer automatisch berechnet und entsprechend auf 0 oder 1 gesetzt. Damit es auch bei einer folgenden Rechnung berücksichtigt werden kann, muß der Programmierer Bit 3 im PSL auf 1 setzen. Vor der ersten Addition setzen wir das CARRY auf 0, und das WITH CARRY auf 1. Die erforderlichen Programmbefehle wurden im vorigen Kapitel schon erläutert, sodaß wir uns der Programmierung der obigen Aufgabe zuwenden können.

Summand A wird mit Hilfe der LODI-Befehle nach R0 und R1 geladen. Das LOW-Byte (1000 0001) kommt nach R0, das HIGH-Byte (0000 0001) nach R1. Mit den ADDI-Befehlen wird jetzt Summand B addiert. Man beginnt immer mit dem LOW-Byte, d.h. der Wert 1000 0101 wird unmittelbar zum Inhalt von R0 addiert. Dann addiert man das HIGH-Byte von B (0000 0001) zum HIGH-Byte von A,das in R1 steht. Bei dieser Addition wird der Übertrag der vorhergehenden Addition automatisch berücksichtigt. Das Additionsergebnis benötigt den Platz von zwei 8 Bit Registern. Es kann also nicht mit einem Ausgabebefehl angezeigt werden. In unserem Beispiel steht das LOW-Byte der Summe in R0, das HIGH-Byte in R1. Wir verwenden die WRTD-Befehle, um die Summe auszulesen. Das Zwischenergebnis müssen wir uns notieren. Aus R0 lesen wir 0000 0110, aus R1 0000 0011 aus. Das Ergebnis entspricht binär dem dezimalen Wert 774.

```
Befehl          * Binärcode * Hexcode * Kommentar

LODI,R0 H08  * 0000 0100 *   04    * Rechnen mit Übertrag vor-
             * 0000 1000 *   08    * schreiben und CARRY-Bit
LPSL         * 1001 0011 *   93    * löschen
LODI,R0 H81  * 0000 0100 *   04    * LOW-Byte von A nach R0
             * 1000 0001 *   81    * laden
LODI,R1 H01  * 0000 0101 *   05    * HIGH-Byte von A nach R1
             * 0000 0001 *   01    * laden
ADDI,R0 H85  * 1000 0100 *   84    * LOW-Byte von B zum
             * 1000 0101 *   85    * LOW-Byte von A addieren
ADDI,R1 H01  * 1000 0101 *   85    * HIGH-Byte von B zum
             * 0000 0001 *   01    * HIGH-Byte von A addieren
WRTD,R0      * 1111 0000 *   F0    * LOW-Byte Summe anzeigen
WRTD,R1      * 1111 0001 *   F1    * HIGH-Byte Summe anzeigen
```

Das Umrechnen eines 16 Bit Wertes in den entsprechenden Dezimalwert ist anfangs nicht ganz leicht. Es soll deshalb noch einmal die dezimale Stellenwertigkeit innerhalb einer großen Dualzahl aufgelistet werden.

```
Bit 0  1     Bit 8  256
Bit 1  2     Bit 9  512
Bit 2  4     Bit 10 1024
Bit 3  8     Bit 11 2048
Bit 4  16    Bit 12 4096
Bit 5  32    Bit 13 8192
Bit 6  64    Bit 14 16384
Bit 7  128   Bit 15 32768
```

Der dezimale Wert der Dualzahl ** 1001 0101 1000 1000 ** ergibt sich aus der Summe 32768 + 4096 + 1024 + 256 + 128 + 8 = 38280.

Bei der Umwandlung einer Dezimalzahl z.B. 45367 in eine Dualzahl kann man mit der SUBTRAKTIONSMETHODE arbeiten:
Man subtrahiert von der Dezimalzahl nacheinander die Wertigkeiten der einzelnen Binärstellen, wobei man mit der höchsten Wertigkeit beginnt. Wenn die Subtraktion ausführbar ist, ohne daß ein negativer Rest besteht, kommt in die betreffende Binärstelle eine 1, ist die Subtraktion nicht ausführbar, wird eine 0 notiert.

```
45367 - 32768 = 12599 > Bit 15 1
12599 - 16384 =-3785  > Bit 14 0
12599 - 8192  = 4407  > Bit 13 1
4407  - 4096  = 311   > Bit 12 1
Wie man leicht sieht, werden die Bits 11, 10 und 9 0.
311   - 256   = 55    > Bit 8 1 ; Bit 7 und 6 werden 0.
55    - 32    = 23    > Bit 5 1
23    - 16    =  7    > Bit 4 1 ; Bit 3 ist 0.
7     - 4     =  3    > Bit 2 1
3     - 2     =  1    > Bit 1 1
1     - 1     =  0    > Bit 0 1
```

Die gesuchte Binärzahl ist ** 1011 0001 0011 0111 **!

RECHNEN MIT BINÄR KODIERTEN DEZIMALZIFFERN (BCD- ARITHMETIK)

Das Prinzip der binär kodierten Dezimalziffern (BCD von engl. binary coded decimal) ist einfach zu verstehen. Die zehn Ziffern von 0 bis 9 müssen verschlüsselt werden, was 4 Bits erfordert:

1 = 0001
2 = 0010
3 = 0011
4 = 0100
5 = 0101
6 = 0110
7 = 0111
8 = 1000
9 = 1001

Vier Bits erzeugen aber 16 mögliche Kombinationen (vergl. mit hexadezimaler Schreibweise) von denen die Kombinationen von 1010 bis 1111 nicht gebraucht werden. Die Dezimalzahl 10 wird im BCD-System nämlich 0001 0000 geschrieben. Der Zahl 23 entspricht 0010 0011. Die größte BCD-Zahl in einem Register ist 99, also 1001 1001.

Durch die unbenutzten Bitkombinationen wird die Arithmetik für die BCD-Zahlen kompliziert. Das Rechenwerk des Mikroprozessors ist es nämlich gewohnt, mit allen möglichen Kombinationen eines 8 Bit-Wertes zu arbeiten.

Mit einigen Beispielen läßt sich das Problem zeigen :

```
  0000 0001 (1)
+ 0000 0011 (3)
  0000 0100 (4)
```

In diesem Fall entsteht ein korrektes Ergebnis. Wir versuchen es einmal mit größeren BCD-Zahlen:

```
  0000 1000 (8)
+ 0000 1001 (9)
  0001 0001 (?)
```

Das Ergebnis lautet 11 in dezimaler Schreibweise: das ist falsch. Das richtige Ergebnis hätte 0001 0111, d.h. 17 sein müssen. Wenn man die Regeln der gewöhnlichen binären Addition anwendet, müssen die **sechs** ungültigen Kombinationen von 1010 bis 1111 übersprungen werden. Mit anderen Worten, liegt das Ergebnis im für BCD-Zahlen verbotenen Bereich, muß pro Halbbyte der Wert 6 hinzugezählt werden.

```
  0000 1000 (8)
+ 0000 1001 (9)
  0001 0001 (11 = falsches Ergebnis)
+ 0000 0110  (6) Korrektur
  0001 0111 (17) richtiges Ergebnis als BCD-Zahl
```

Bei der BCD-Rechnung muß also immer dann sechs zum Resultat hinzuaddiert werden, wenn die Addition durch den verbotenen Bereich oder in ihn hineinführt. Welche Programmbefehle ermöglichen diese Dezimalkorrektur?

Der 2650 hat für die BCD-Rechnung einen besonderen Befehl mit der Abkürzung **DAR,r** (engl. decimal adjust register). Für seine Anwendung gelten besondere Regeln:

Bei der **ADDITION** ist zunächst der Wert H66 zur ersten BCD-Zahl zu addieren. Es wird also so getan, als ob die folgende Addition zwangsläufig in den verbotenen Bereich hineinführt und deshalb der Wert jedes Halbbytes durch Addition von 6 zu korrigieren wäre. Dann wird die zweite BCD-Zahl addiert und als nächstes der DAR-Befehl mit der Summe durchgeführt. Durch den DAR-Befehl prüft der Rechner ob die vorsorgliche Addition von 66 ganz oder teilweise zurückgenommen werden muß. Die Bedingung liefern das CARRY und das sogenannte INTERDIGIT CARRY (IDC). Das IDC ist das 5.Bit im unteren Programm-Status-Wort, es speichert einen Übertrag von Bit 3 nach Bit 4 in einem Rechenregister. Bei unserem zweiten Beispiel auf der vorigen Seite entstand ein solcher Halbbyte-Übertrag. Bei einem gesetzten IDC erkennt der Prozessor, daß die vorsorgliche Addition von 6 zum unteren Halbbyte zu Recht erfolgte und nicht zurückgenommen werden darf. Wurde das IDC nicht gesetzt, subtrahiert der intelligente Prozessor 6 vom unteren Halbbyte. Die gleiche Prüfung führt der Prozessor mit dem oberen Halbbyte durch, diesmal wird allerdings das CARRY auf 0 oder 1 geprüft und die Addition von 6 zurückgenommen, wenn es **nicht** gesetzt wurde.

Bei der **SUBTRAKTION** sind die Programmschritte kürzer. Wenn beide Zahlen - Subtrahend und Minuend - als BCD-Zahlen vorgegeben waren, muß lediglich mit der Differenz der DAR-Befehl durchgeführt werden. Das Ergebnis ist immer eine BCD-Zahl.

Bei der Programmierung von BCD-Rechnungen muß im unteren PSW immer die Arbeit mit CARRY vorgeschrieben werden und das CARRY (Bit 0) und das INTERDIGIT CARRY (Bit 5) richtig gesetzt sein.
Der DAR-Befehl ist im Begleitbuch auf S.351 beschrieben:

```
DAR,R0   1001 0100   94 (hex)
DAR,R1   1001 0101   95 (hex)
DAR,R2   1001 0110   96 (hex)
DAR,R3   1001 0111   97 (hex)
```

Mit den Befehlen zur BCD-Arithmetik kann man ebenfalls Zahlen verarbeiten, die den 8 Bit Bereich überschreiten. Will man z.B. die Addition 1256 + 6733 durchführen, müssen in einem ersten Schritt die Einer und Zehner der Summanden, dann die Hunderter und Tausender addiert werden. Unter Einbeziehung des CARRYs bei der jeweils folgenden Rechnung erhalten wir ein richtiges Gesamtergebnis. Auf der nächsten Seite finden Sie 3 Beispiele.

BEISPIEL: 1 34 + 17 im BCD-System

```
ADR BINÄRCODE  HEX  ASSEMBLER     * KOMMENTAR
00  0000 0100   04  LODI,R0 08    * Rechnen mit Übertrag
01  0000 1000   08                 * vorschreiben; C = 0
02  1001 0011   93  LPSL           *
03  0000 0100   04  LODI,R0 34    * 1. BCD-Summanden
04  0011 0100   34                 * laden
05  1000 0100   84  ADDI,R0 66    * wegen BCD-Arithmetik
06  0110 0110   66                 * H66 addieren
07  1000 0100   84  ADDI,R0 17    * 2. BCD-Summanden
08  0001 0111   17                 * addieren
09  1001 0100   94  DAR,R0         * Dezimalkorrektur der Summe
0A  1111 0000   F0  WRTD,R0        * Summe ausgeben
```

BEISPIEL: 2 56 - 23 im BCD-System

```
00  0000 0100   04  LODI,R0 09     * Rechnen mit Übertrag
01  0000 1001   09                 * vorschreiben; C = 1
02  1001 0011   93  LPSL           * wegen Subtraktion
03  0000 0101   05  LODI,R1 56     * 56 laden
04  0101 0110   56                 *
05  1010 0101   A5  SUBI,R1 23     * 23 subtrahieren
06  0010 0011   23                 *
07  1001 0101   95  DAR,R1         * Dezimalkorrektur
08  1111 0001   F1  WRTD,R1        * Differenz ausgeben
```

BEISPIEL: 3 1278 + 3123 im BCD-System

```
00  0000 0100   04  LODI,R0 08     * Rechnen mit Übertrag
01  0000 1000   08                 * vorschreiben; C = 0
02  1001 0011   93  LPSL           *
03  0000 0100   04  LODI,R0 78     * Zehner und Einer
04  0111 1000   78                 * laden
05  1000 0100   84  ADDI,R0 66     * 66 addieren
06  0110 0110   66                 *
07  1000 0100   84  ADDI,R0 23     * Zehner und Einer der
08  0010 0011   23                 * 2.Zahl addieren
09  1001 0100   94  DAR,R0         * Dezimalkorrektur
0A  0000 0101   05  LODI,R1 12     * Hunderter und Tausender
0B  0001 0010   12                 * laden
0C  1000 0101   85  ADDI,R1 66     * 66 addieren
0D  0110 0110   66                 *
0E  1000 0101   85  ADDI,R1 31     * Hunderter und Tausender
0F  0011 0001   31                 * der 2. Zahl addieren
10  1001 0101   95  DAR,R1         * Dezimalkorrektur
11  1111 0000   F0  WRTD,R0        * Zehner und Einer, dann
12  1111 0001   F1  WRTD,R1        * Hunderter und Tausender
                                   * der Summe ausgeben
```

Das Beispiel wurde so gewählt, daß im High-Byte der Summe der Wert 99 nicht überschritten wird.