PDP-11 Maschinensprache

G 582-1

INHALTSVERZEICHNIS

KAPITEL 1 - EINFÜHRUNG

1.1 PDP 11 Familie

(PDP = Programmable Data Processor)

Die PDP-11 Familie beinhaltet mehrere 16-Bit-Zentraleinheiten, eine große Anzahl von peripheren Geräten und Zusätzen sowie eine ausbaufähige Software. PDP-11 Maschinen sind von der Architektur her ähnlich, Hardware und Software sind aufwärts kompatibel. Andererseits hat jedoch jede Maschine ihre eigene Charakteristik. Neue PDP-11 Systeme werden mit den bereits bestehenden Mitgliedern dieser Familie kompatibel sein. Der Benutzer kann das System aussuchen, welches für seine Anwendungen als günstigstes erscheint. Sollten jedoch die Erfordernisse wachsen oder sich gar ändern, so kann die Hardware auf einfachste Weise erweitert oder geändert werden. Die wesentlichsten Eigenschaften der PDP-11 Systeme sind in der Tabelle 1-1 zusammengestellt.

Von den PDP-11 Prozessrechnern werden folgende Typen angeboten:

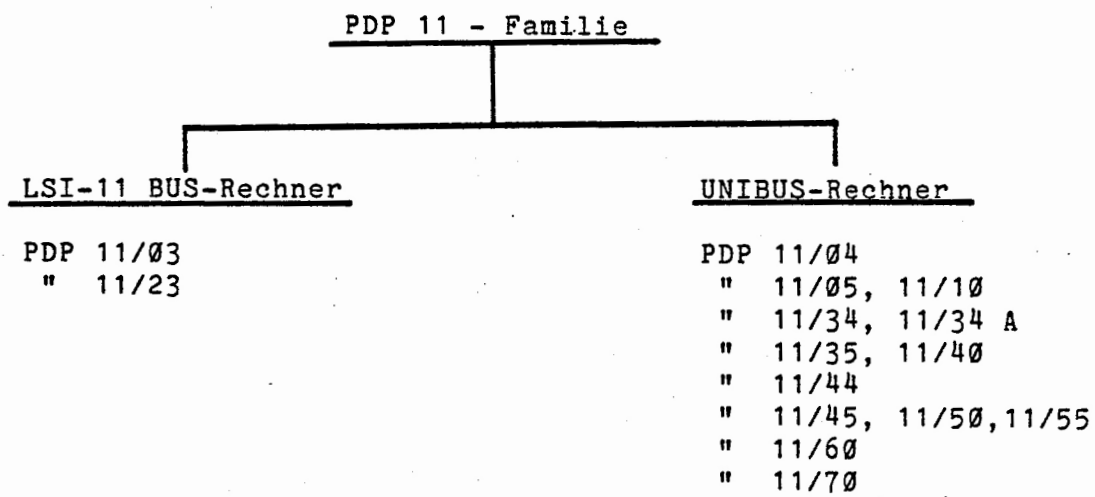
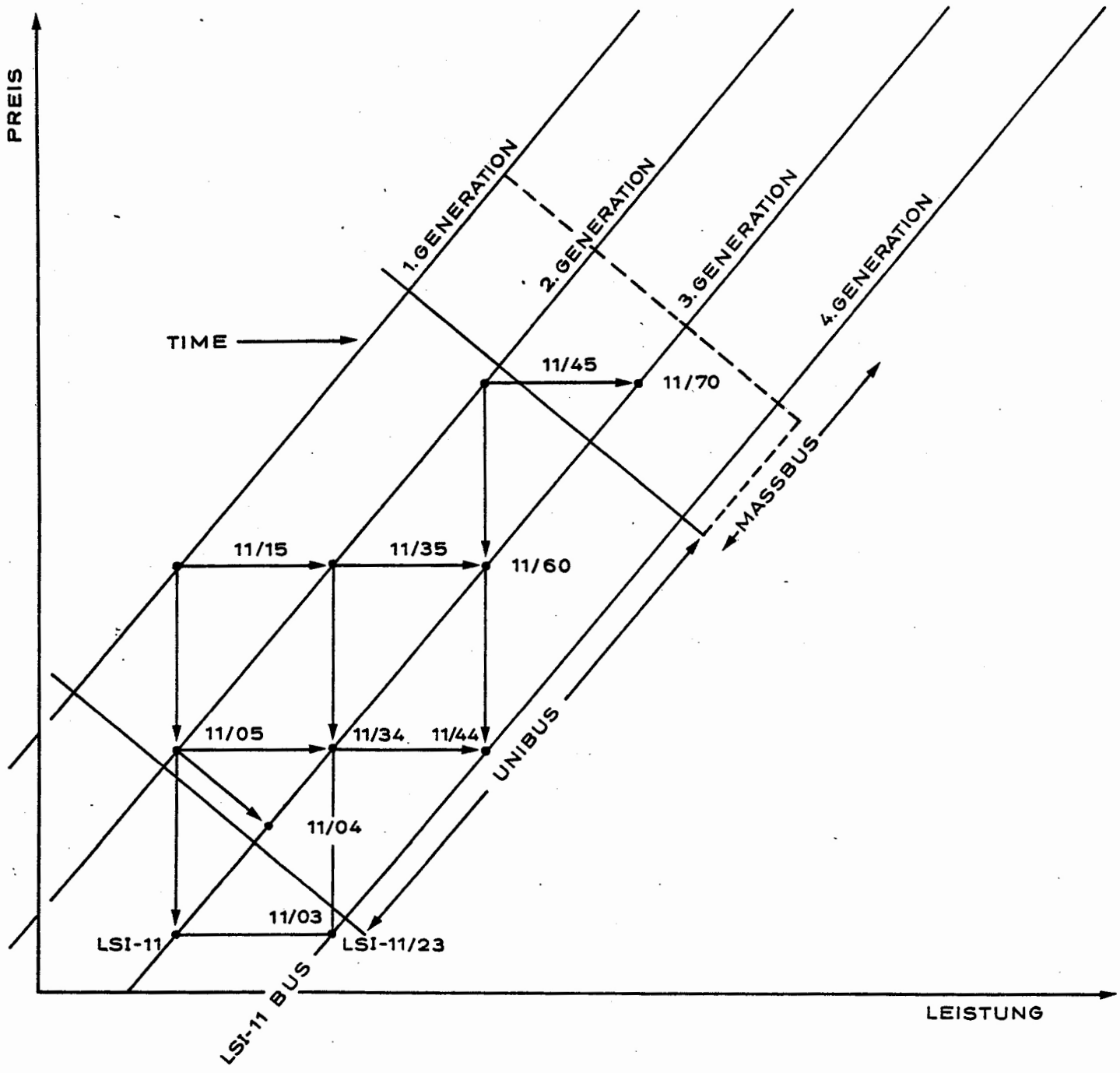


Tabelle 1.1

CPU	03 LSI-11	04	05 (OEM)	10 (END)	23	34	35 (OEM)	40 (END)	44	45	50	55	60	70
max.Adr. Bereich(KW)	32	32	32	32	128	128	128	128	2048	128	128	128	128	2048
BUS	LSI-11 Bus	U	U	U	LSI-11 Bus	U	U	U	U M.Bus	U + Fastbus			U	U M.Bus
Interruptebenen HW SW	1	4	4	4	4	4	4	4	4 7	4 7	4 - 7	4 7	4 7	4 7
Betr.Arten Kernel Superv User	x	x	x	x	x x x	x x	x x	x x	x x x	x x x	x x x	x x x	x x	x x x
GRPs	8	8	8	8	8	8	8	8	10	16	16	16	9	16
Progr.Console Oper. Console	x LSI-o.	Opt. x	x	x	x	Opt. x	x	x	Elec. Cons.	x	x	x	x	x Elec. Cons-
Memory Managem. Cache					x	x Opt.	Opt.	Opt.	x 4Kw	Opt.	Opt.	x	x 1Kw	x 1Kw
FP					Opt.	Opt. (34A)	Opt.		Opt.	Opt.	Opt.	Opt.		Opt.
EIS CIS					x	x	Opt.	Opt.	x Opt.	x	x	x		x
Memory	Core MOS PROM	Core MOS	Core	Core	NOS	Core MOS	Core	Core	MOS	Core	MOS	BIP		Core ECC-MOS
Large Instr. Set									x	x	x	x	µ progr.	x



Preis / Leistungsvergleich

1.2 Generelle Eigenschaften

1.2.1 Der UNIBUS (Bild 1.1.)

Alle Komponenten einer PDP11 Maschine (Zentraleinheit, Speicher, Peripherie) sind an einen gemeinsamen Kanal -den UNIBUS- angeschlossen. Über ihn läuft die gesamte Kommunikation d. h. er überträgt alle Daten, Adressen und Steuersignale.
Durch die bidirektionale Verwendbarkeit des UNIBUS kann ein Gerät Informationen senden und empfangen und Daten ohne Mitwirkung der Zentraleinheit verarbeiten. Zudem ist der UNIBUS asynchron. Das bedeutet, daß jedes Gerät mit der ihm eigenen Geschwindigkeit tätig sein kann und die Systemleistung insgesamt verbessert wird.

1.2.2 Der LSI-11-BUS

Für die Rechner PDP 11/Ø3, 11/23 wurde eine eigene Busstruktur entwickelt: der LSI-11-BUS.

Im Gegensatz zum UNIBUS werden während des Betriebs Adressen und Daten im Zeitmultiplex übertragen. Die Zentraleinheit bringt zuerst die Adresse der gewünschten Speicherstelle oder des I/O-Interfaces auf den Bus, es folgen Steuersignale, die die Gültigkeit der Adresse bestätigen. Da der Bus bi-direktional ist, zeigen weitere Steuersignale die Richtung des Datenflusses an. Das adressierte Modul übersetzt diese Signale und reagiert, indem er die Daten entweder von der Zentraleinheit akzeptiert oder ihr Daten zuführt.

1.2.3 Interrupt (Unterbrechungen)

Ein automatisches Interrupt System auf Prioritätsbasis erlaubt der Zentraleinheit, auf Anforderungen außerhalb des Systems oder in der Zentraleinheit selbst automatisch zu reagieren. Jede beliebige Anzahl von Geräten kann auf verschiedenen Ebenen angeschlossen werden. Jedes periphere Gerät innerhalb des PDP-11 Systems hat einen "Hardware-Vector" (Zeiger) zu zwei eigens diesem Gerät zugeordneten Arbeitsspeicher-Worten. Ein Wort zeigt auf die Service-Routine für dieses Gerät und das andere beinhaltet eine Status-Information für die Service-Routine. Diese einheitliche Identifikation erübrigt das Ablegen der Geräteadressen in einer Liste, da die Hardware die Auswahl und Durchführung der entsprechenden Service-Routine vornimmt, sobald der Status des unterbrochenen Programms automatisch gerettet wurde.

1.2.4 DMA (Direkter Speicherzugriff)

Alle PDP-11 Systeme ermöglichen den direkten Speicherzugriff. Wir bezeichnen ihn nachfolgend als DMA (Direkt Memory Acess). Jede beliebige Anzahl von DMA-Geräten kann an den UNIBUS bzw. LSI-11-BUS angeschlossen werden. Einem DMA-Gerät wird die höchste Priorität zugeordnet, um jederzeit Daten lesen und schreiben zu können. Die Verlustzeit ist auf Grund der Organisation und Logik des UNIBUS auf ein Minimum reduziert.

Alle Steuerleitungen für Interrupt und DMA sind im UNIBUS bzw. LSI-11-BUS integriert.

1.2.5 CPU (Zentraleinheit)

Die Zentraleinheit, als Untersystem an den UNIBUS angeschlossen, überwacht die Zuordnung des UNIBUS für die Peripherie und führt neben arithmetischen und logischen Operationen die Befehlsdekodierung durch. Sie beinhaltet mehrere schnelle, allgemein verwendbare Register, die als Akkumulatoren, Zeiger, Index-Register oder als Zeiger für automatischen Index-Modus benutzt werden können. Die Zentraleinheit führt den direkten Transfer zwischen Ein/Ausgabegeräten und dem Arbeitsspeicher durch, ohne dabei den Inhalt der Register zu zerstören; bearbeitet 16-bit-Wort Daten ebenso wie 8-bit-Byte Daten und -unter Verwendung der dynamischen "STACK" -Technik- ist die Möglichkeit von verschachtelten Interrupts (Unterbrechungen) und Subroutinen (Unterprogrammen) gegeben.

Befehls-Struktur

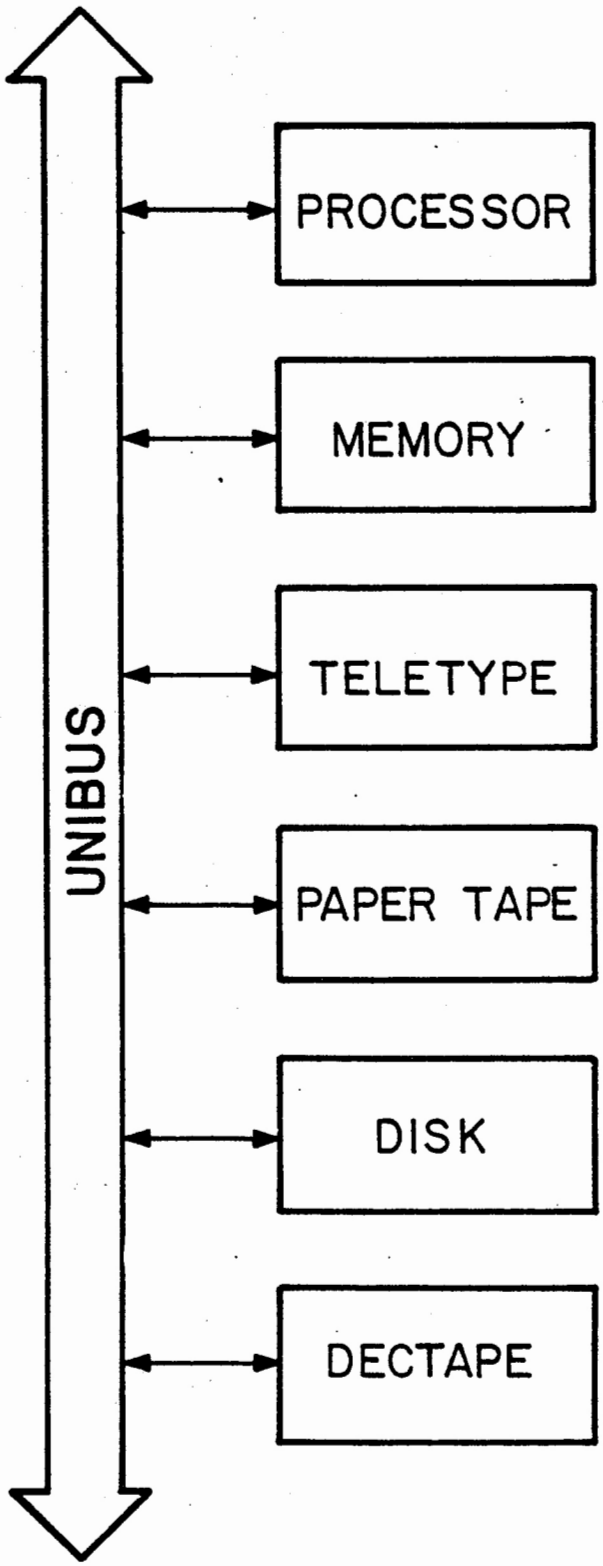
Der Befehlssatz verwendet die Flexibilität der generellen Register, um daraus über 400 verdrahtete Befehle zu erstellen -das leistungsfähigste Befehlsrepertoire eines Rechners der 16-bit Klasse überhaupt. Im Gegensatz zu konventionellen 16-bit Rechnern, die normalerweise drei Klassen von Befehlen haben (Arbeitsspeicher bezogene Befehle, Operations-Befehle und Ein/Ausgabe Befehle), werden in der PDP-11 alle Operationen durch einen einzigen Befehlssatz ausgeführt. Da die Register von peripheren Geräten genauso flexibel behandelt werden können wie z. B. der Arbeitsspeicher durch die Zentraleinheit, können also die Befehle gleichermaßen für die Manipulation von Arbeitsspeicherdaten wie auch für die Register der peripheren Geräte verwendet werden. Zum Beispiel können Daten eines externen Geräte-Registers direkt durch die Zentraleinheit getestet oder modifiziert werden, ohne sie dabei in den Arbeitsspeicher zu bringen oder die Inhalte der Prozessor-Register zu zerstören.

1.2.6 Memory (Arbeitsspeicher)

Arbeitsspeicher verschiedenster Geschwindigkeit und Charakteristika können in einem einzigen PDP-11 System frei verwendet und untereinander ausgetauscht werden. Wachsen also die Erfordernisse an Speichergröße oder Speicher-Technologie, ist damit keine Veralterung wie bei herkömmlichen Rechnern gegeben.

Bisher zur Verfügung stehende Speicher sind in Magnetkern-, MOS- und Bipolarer Technik ausgeführt.

Tabelle 1.1 gibt Auskunft über Speichergrößen.



SYSTEM BLOCK DIAGRAM

Bild 1.1

1.3 Periphere Geräte

Über den UNIBUS bzw. LSI-11-BUS sind eine große Anzahl von peripheren Geräten an die Zentraleinheiten der PDP 11 - Familien anschließbar.

1.3.1 Ein/Ausgabegeräte

Alle PDP-11 Systeme stehen mit DECterminals oder DECwriter als Standard-Einheit zur Verfügung. Jedoch können die Ein/Ausgabe-Möglichkeiten um schnelle Lochstreifenleser und -stanzer, Zeilendrucker, Kartenleser oder alpha-numerische Sichtgeräte erweitert werden. Der "DECwriter" -ein komplett von DEC entwickeltes und hergestelltes Gerät hat den Standard-Fernschreiber (TELETYPE genannt) abgelöst. Die Vorteile der neuen Terminals gegenüber den üblichen elektromechanischen Fernschreibern sind u. a. höhere Geschwindigkeit, weniger mechanische Teile und Geräuscharmut.

Unter die PDP-11 Ein/Ausgabegeräte fallen:

DECterminal alphanumerische Sichtgeräte:
VT 50, VT 52, VT 100 usw.

DECwriter Fernschreiber:
LA 30, LA 34, LA 36, LA 38, LA 120

Schnelle Zeilendrucker:
LP Ø5, LP 25 usw.

Schnelle Lochstreifenleser und -stanzer
PC 11

Fernschreiber (Teletype)
ASR 33

Kartenleser
CD 11

1.3.2 Speicher-Geräte

Die Speichergeräte reichen von bequemen kleinen Magnetband-Einheiten bis zu großen Magnetbändern und Plattenspeichern. Über den UNIBUS kann eine große Anzahl von Speichergeräten der verschiedensten Kombinationen an das PDP-11 System angeschlossen werden.

Magnetbandeinheiten: TE 10, TE 16
TS Ø3, TS 11
TU 45, TU 56, TU 58, TU 60, TU 77 usw.

Plattenspeicher: RK Ø5, RL Ø1, RL Ø2
RX Ø1, RX Ø2
RM Ø2/Ø3
RP Ø4/Ø5/Ø6 usw.

Die Kapazitäten dieser (Massen- oder Hintergrund-) Speicher-Geräte reichen bis zu 200 Millionen Bytes und mehr.

1.4 PDP 11- Software

Für die Familie der PDP-11 Systeme ist eine erweiterungsfähige Software für Platten- und Lochstreifensysteme erhältlich. Je größer die PDP-11 Konfiguration ist, desto umfangreicher und vorteilhafter wird die mitgelieferte Software.

1.4.1 Betriebssysteme (Operating Systems)

Unter den Betriebssystemen einer Datenverarbeitungsanlage wird eine, durch ihre Ausstattung festgelegte Menge von Organisationsprogrammen und Programmsystemen verstanden. Diese sind vom speziellen Anwendungsfall unabhängig und werden in der Regel vom Hersteller geliefert. Mit dem Betriebssystem wird die technische Struktur eines Rechners um die programmierte Struktur erweitert,so daß der Benutzer die Rechenanlage mit möglichst bequemen Mitteln an seine speziellen Aufgaben anpassen kann. Man könnte das Betriebssystem im übertragenen Sinn als Dirigent oder Überwacher bezeichnen, der alle Aufgaben und Operationen koordiniert. Anlagenausstattung und Benutzerwünsche bestimmen die Varianten der Betriebssysteme.

Aufgaben des Betriebssystems

Die erste Grundfunktion eines Betriebssystems ist es, eine DVA "bedienbar" zu machen, d. h. dem Benutzer den Verkehr mit der Rechenanlage überhaupt erst zu ermöglichen. Unter "bedienen" ist zu verstehen, daß der Benutzer durch Eingabe von alphanumerischem Text oder durch Betätigung von Funktionstasten die Anlage zu bestimmten, gewünschten Reaktionen veranlassen kann (Kommandos, Bedienungsanweisungen). Die einfachsten Bedienungsfunktionen sind:

Eingabe und Start von Programmen

Die Eingabe erfolgt entweder von den "klassischen" Datenträgern, Lochkarte bzw. Lochstreifen, oder von magnetischen Datenträgern wie Magnetbänder, Trommel- oder Plattenspeichern. Den Transfer in den Arbeitsspeicher des Rechners bezeichnet man als Laden des Programms. Dieses Laden übernimmt z. B. das Betriebssystem. Die Funktion "Programm laden" kann bei den verschiedenen Betriebssystemen mit mehr oder weniger Bedienungskomfort ausgestattet sein.

Einige Betriebssysteme

RT11	Hier handelt es sich um ein plattenorientiertes Einzelbenutzersystem; für die Programmentwicklung mit einer Background-Foreground-Version, die Echtzeitprogramme im Foreground unterstützen kann und gleichzeitig Programmentwicklung im "Background" ermöglicht.
RSTS/E	Ein umfangreiches Timesharing-Betriebssystem für die PDP-11, das simultan bis zu 32 Benutzer bedienen kann.
RSX-11	Diese Echtzeitbetriebssysteme können zahlreiche Aufgaben gleichzeitig ausführen.
IAS	Dieses Betriebssystem findet bei großen PDP-11-Konfigurationen Einsatz. Es kann Timesharing-, Batch- und Echtzeitanwendung geichzeitig fahren.
MUMPS-11	Ein flexibles und dynamisches Betriebssystem für den Aufbau und die Verwaltung von umfangreichen Datenbanken.

1.4.2 Testprogramme

Für den Test einzelner Hardware-Komponenten oder beliebiger System-Konfigurationen stehen vom Betriebssystem unabhängige Testprogramme zur Verfügung. Sie erleichtern ganz entscheidend die Fehlersuche und verkürzen entsprechend ev. anfallende Reperaturzeiten.

Oberbegriff für Testsoftware: XXDP+ Diagnostic Software

1.5 Darstellung von Zeichen und Ziffern

Im folgenden wird das Zahlensystem und die Darstellung von Ziffern und Zeichen in einer PDP 11 festgelegt.

1.5.1 Bit, Byte, Wort

In einem Programm kann ein Benutzer Bits, Bytes und Worte direkt ansprechen.

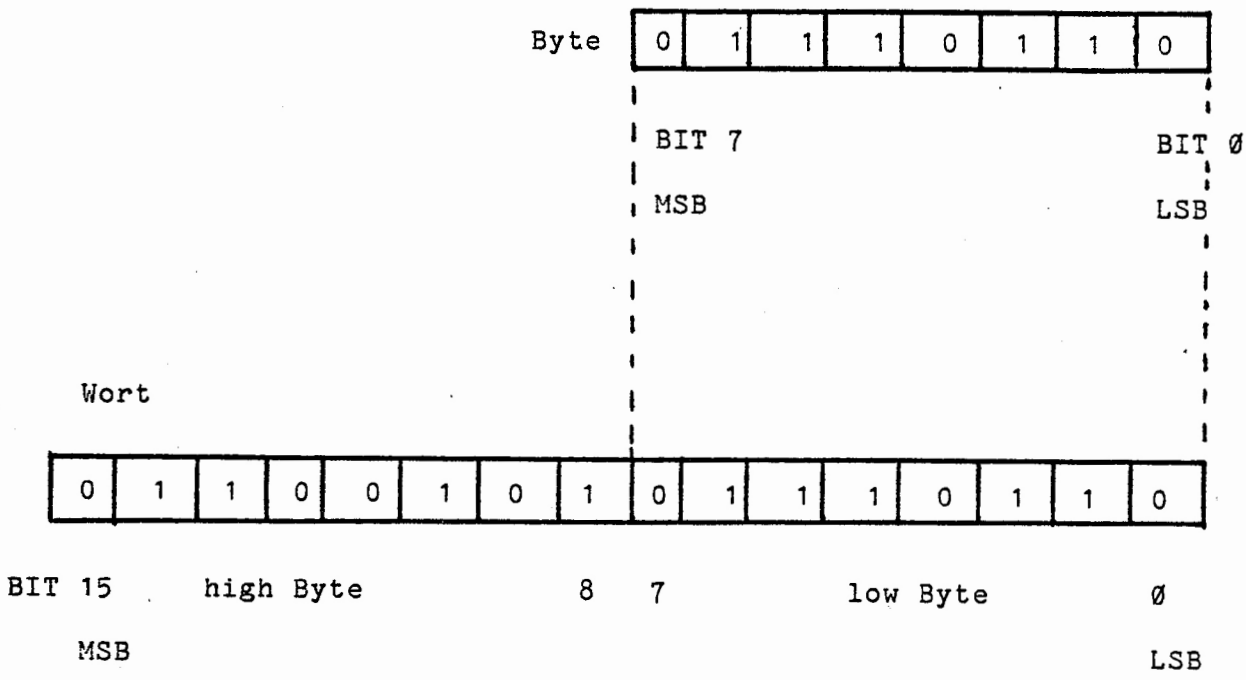
Definition:
Ein BIT ist die kleinste ansprechbare Einheit.

Ein BYTE besteht aus 8 Bits.

Ein WORT besteht aus 16 Bits oder 2 Bytes, high Byte und low Byte.

LSB = Least Significant Bit - Bezeichnung für das Bit mit der niedrigsten Wertigkeit
MSB = Most Significant Bit - Bezeichnung für das Bit mit der höchsten Wertigkeit

Beispiel



1.5.2 Zahlen- und Zeichendarstellung

Der gesamte Zahlen- und Zeichenvorrat läßt sich in Bits, Bytes und Worten darstellen. Zur vereinfachten Schreibweise werden je 3 Bits zusammengefaßt. Aus diesem Grunde rechnet man mit Oktalzahlen ($2^3 = 8$).

Eine OKTALZIFFER ist also in 3 Bits darstellbar.
Zahlenbereich einer Oktalziffer: Ø...7

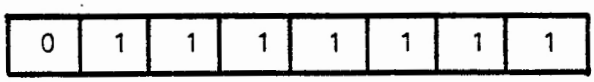
Buchstaben, Ziffern und Sonderzeichen lassen sich auch durch eine 7-Bit Kodierung darstellen. Eine spezielle Kodierung ist der ASCII-Code.

EINHEIT	1 BIT	1 BYTE	1 WORT
INHALT	0 od. 1	8 Bits	16 Bits bzw. 2 Bytes
POSITIVER ZAHLEN BEREICH	-	$Ø...177_8$ bzw. $Ø...+127_{10}$	$Ø...77777_8$ bzw. $Ø...+32767_{10}$
NEGATIVER ZAHLEN BEREICH	-	$377_8...200_8$ bzw. $-1_{10}...-128_{10}$	$177777_8...100\ 000_8$ bzw. $-1_{10}...32768_{10}$
ASCII	-	1 ASCII-Zeichen und ein Paritätsbit	2 ASCII-Zeichen und je ein Paritätsbit

Negative Zahlen werden im sogenannten Zweier-Komplement dargestellt.

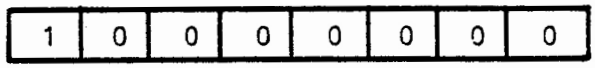
Beispiele zur Zahlen- und Zeichendarstellung

Byte
- Zahlendarstellung



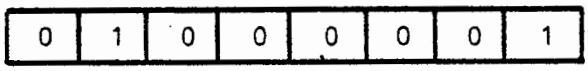
177_8 oder + 127_{10}

Bit 7 = Vorzeichenbit



200_8 oder -128_{10}

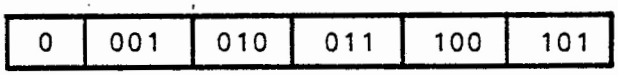
- Zeichendarstellung (ASCII)



101_8 ASCII:A

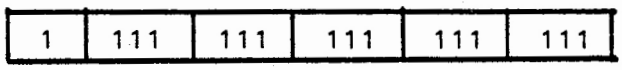
Bit 7 = Paritätsbit

Wort
- Zahlendarstellung



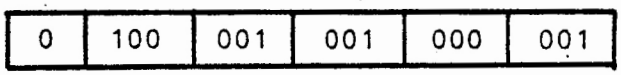
$+12345_8$

Bit 15 = Vorzeichenbit



177777_8 oder -1_{10}

- Zeichendarstellung



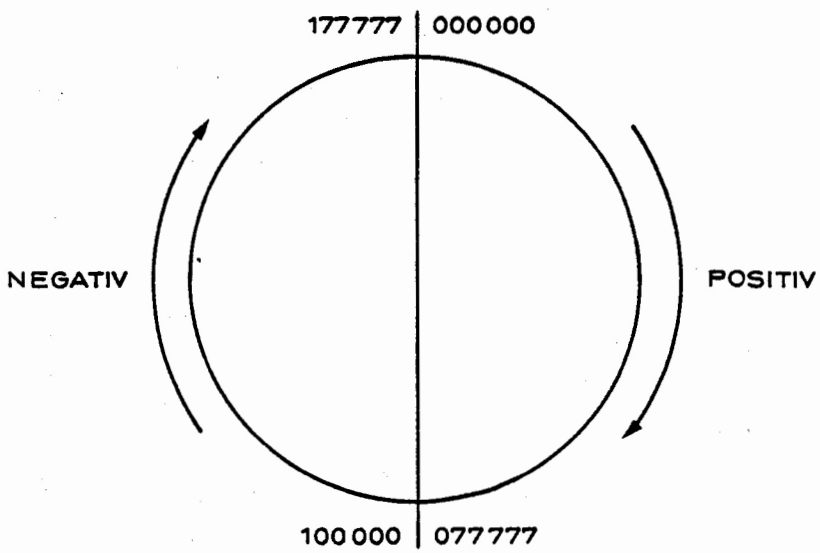
Bit 15 u. Bit 7 : Paritätsbit
Bits 8...14 : 102_8=ASCII: B
Bits Ø...6: 101_8=ASCII: A

Anmerkung

Man kann dem Inhalt eines Speichers nicht ansehen, ob Zahlen, Adressen, Befehle oder ASCII-Zeichen abgespeichert wurden. Diese Interpretation bleibt dem Programmierer überlassen.

1.5.3 Graphische Darstellung der positiven und negativen Zahlen (Zahlenkreis)

1. Wortformat



	dezimal	oktal
höchste positive Zahl:	+ 32767	0 77 777
	+ 32766	0 77 776
	.	
	.	
	.	
	+ 1	0 00 001
	0	0 00 000
	- 1	1 77 777
	- 2	1 77 776
	.	
	.	
	.	
	- 32767	1 00 001
höchste negative Zahl:	- 32768	1 00 000

Arbeitsblatt 1.1

1. Wie groß ist der Zahlenbereich eines Bytes (8 Bit)?
   Benützen Sie Hilfsblatt 1

2. Gegeben sei die Bitkombination

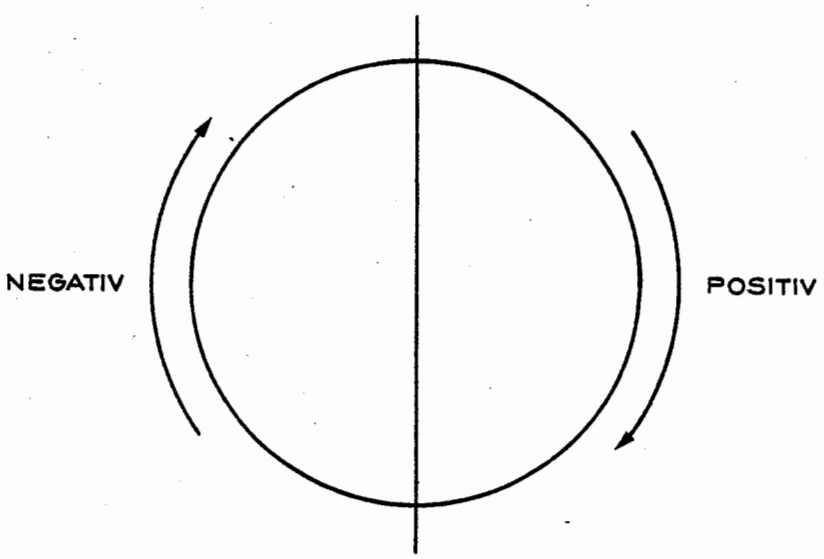
0	1	0	0	0	0	0	1	0	1	0	1	0	0	1	0

   Interpretieren Sie diese Kombination als

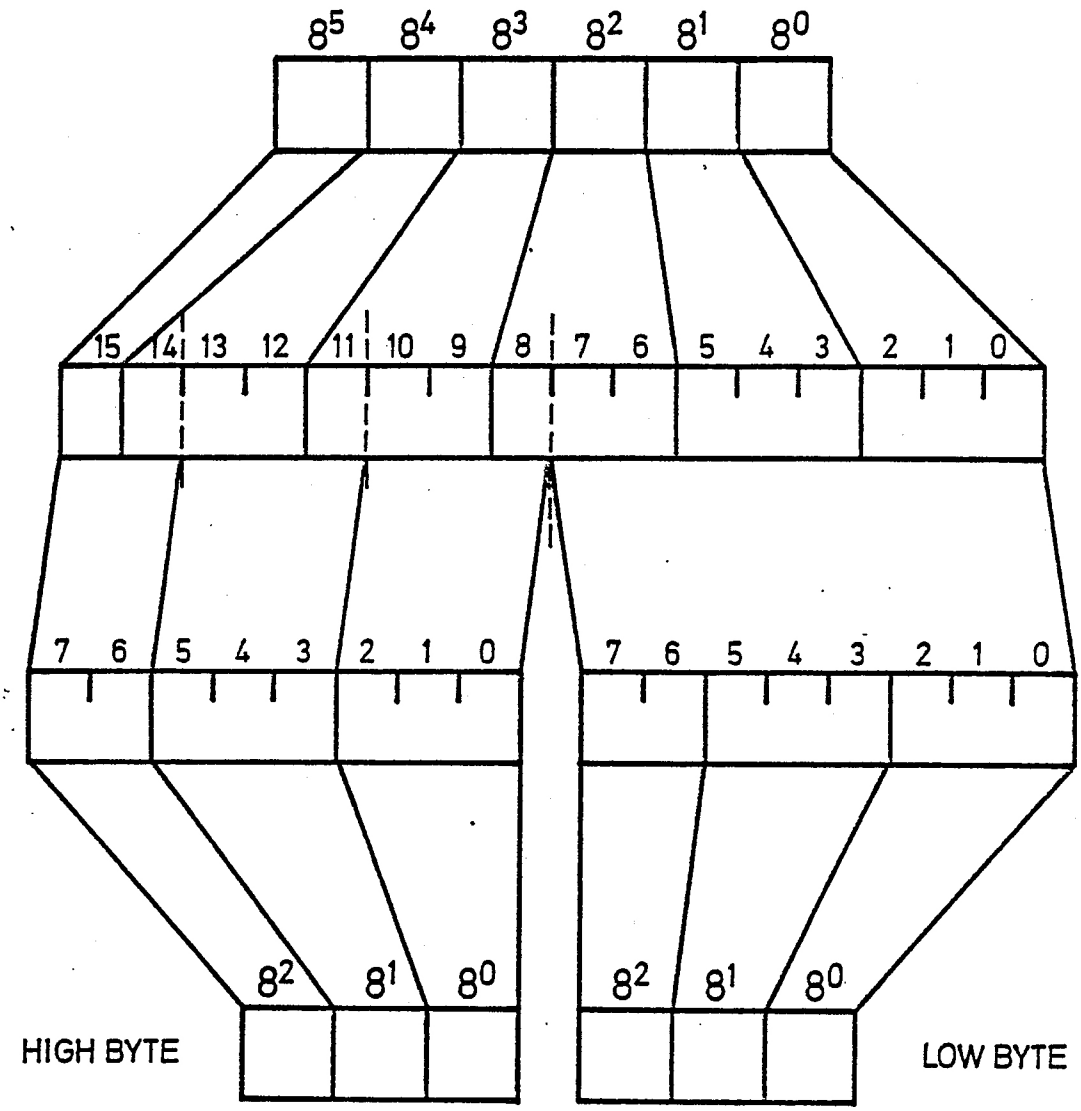
   a) arithmetische 16 Bit Zahl
   b) arithmetische 8 Bit Zahlen
   c) ASCII-Zeichen
   d) Befehl

   Verwenden Sie Hilfsblatt 2

Hilfsblatt 1



Hilfsblatt 2



KAPITEL 2 - SYSTEMAUFBAU

2.1 Systemüberblick

Bild 2.1 gibt einen guten Überblick über eine System-Konfiguration und stellt die zentrale Bedeutung des UNIBUS heraus.

Mit Ausnahme des Speichers (Memory) kann jedes Gerät den UNIBUS anfordern (Request), um Informationen zu einem anderen Gerät (Device) zu übertragen. Alle Geräteregister-Adressen liegen in den obersten 4K des adressierbaren Bereichs. Auf Grund dieser Struktur kann die Zentraleinheit (CPU) Geräteregister wie Speicherzellen behandeln und die gleichen Befehle darauf anwenden.

Der UNIBUS stellt die Verbindungswege für Adressen, Daten und Steuer-Informationen für alle Geräte in bidirektionaler Richtung her. Daraus resultiert, daß ein Geräteregister für Ein- und Ausgabe verwendet werden kann.

Die Geräte-Kommunikation über den UNIBUS erfolgt nach einem sogenannten "MASTER-SLAVE" (Herr und Sklave)-Prinzip. Zum Zeitpunkt einer BUS-Operation hat ein Gerät die Kontrolle über den BUS. Dieses Gerät -als "MASTER" bezeichnet- kontrolliert den BUS, wenn es mit einem anderen Gerät -als "SLAVE" bezeichnet- in Verbindung tritt.

Das o. g. Prinzip ist dynamisch: So kann z. B. die Zentraleinheit als Master Steuer-Informationen zur Platte (SLAVE) schicken,die dann wiederum den BUS überwacht (als MASTER), wenn sie mit dem Speicher in Verbindung tritt.

Der UNIBUS wird von der Zentraleinheit und von allen anderen Geräten verwendet. Eine Prioritäts-Struktur entscheidet, welches Gerät die Kontrolle über den BUS bekommt. Somit ist also jedem Gerät, welches Master für den BUS werden kann, eine gewisse Priorität zugeordnet. Wenn zwei Geräte zur gleichen Zeit nach dem BUS verlangen, übernimmt das Gerät mit der höheren Priorität die Kontrolle.

Die Kommunikation zweier Geräte wird nach dem sogenannten "handshaking"-Verfahren durchgeführt: Für jedes vom Master gesendete Steuer-Signal wird vom Slave eine Rückantwort gegeben. Dadurch ist die Kommunikation unabhängig von der BUSLänge oder der Geschwindigkeit, mit der ein Gerät arbeitet.

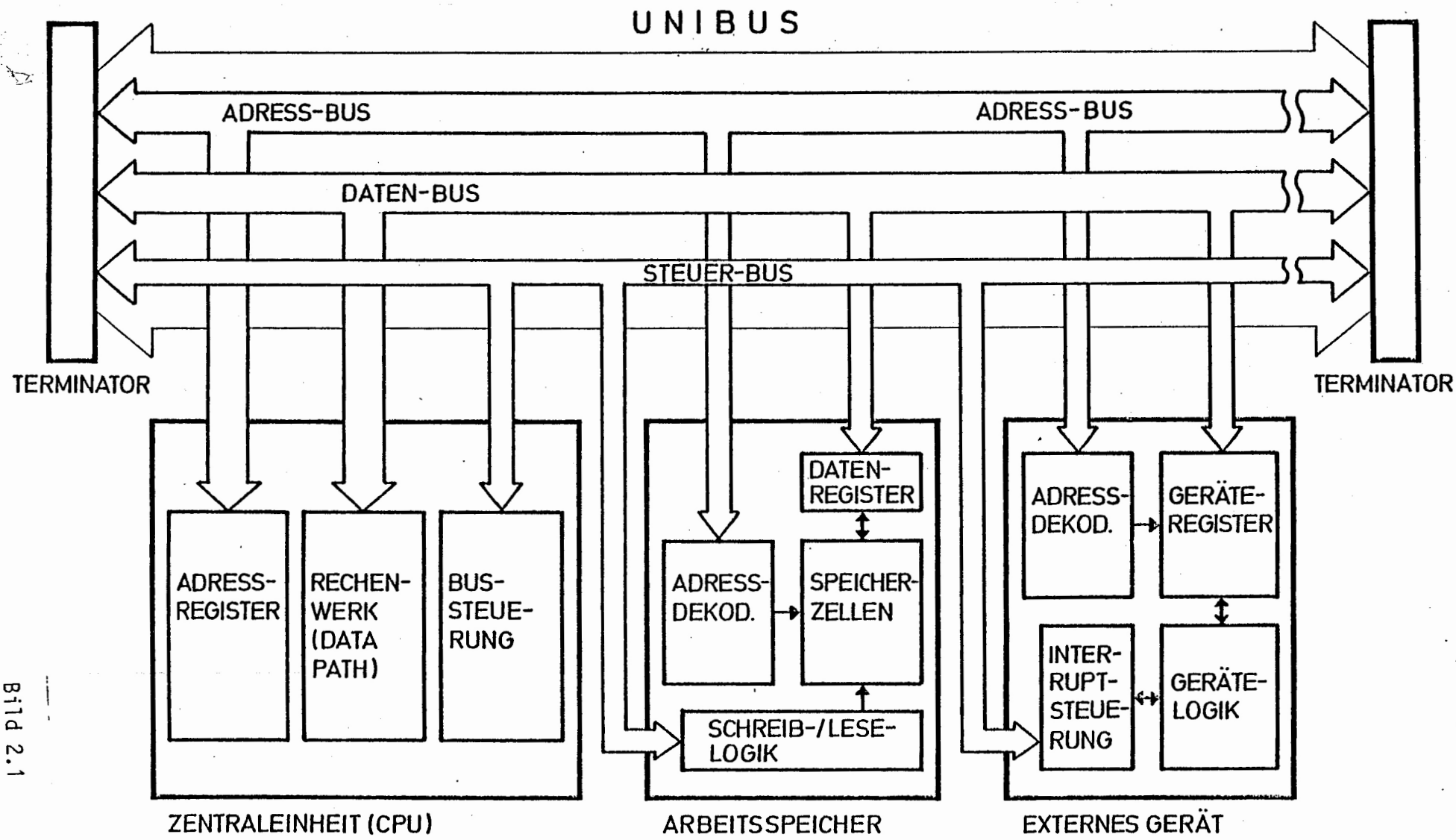

UNIBUS
ADRESS-BUS
ADRESS-BUS
DATEN-BUS
STEUER-BUS
TERMINATOR
TERMINATOR
ADRESS-REGISTER
RECHEN-WERK (DATA PATH)
BUS-STEUE-RUNG
ZENTRALEINHEIT (CPU)
DATEN-REGISTER
ADRESS-DEKOD.
SPEICHER-ZELLEN
SCHREIB-/LESE-LOGIK
ARBEITSSPEICHER
ADRESS-DEKOD.
GERÄTE-REGISTER
INTER-RUPT-STEUE-RUNG
GERÄTE-LOGIK
EXTERNES GERÄT

Bild 2.1

2.2 CPU (Zentraleinheit)

2.2.1 Aufgaben

Die CPU führt alle vom System verlangten arithmetischen und logischen Operationen durch und verarbeitet eine große Anzahl von Sprung- und Testbefehlen (Instruktions-Decorder, ALU, Prozessor-Statuswort-PSW)

Weiterhin entscheidet sie, wer die UNIBUS-Kontrolle übernehmen soll und überträgt diese Kontrolle dann dem Gerät mit der höchsten Priorität (UNIBUS-Interface und Steuerung)

2.2.2 Aufbau (Bild 2.2)

1. Unibus-Interface

Das UNIBUS-Interface sorgt für die Anpassung des Prozessor an den UNIBUS. Alle Signale werden mit BUS-Treibern auf den BUS gegeben bzw. mit BUS-Empfängern vom BUS abgegriffen. Ferner sorgt das UNIBUS-Interface für die zeitliche Steuerung (Timing) aller BUS-Signale.

2. Data Path (Rechenwerk)

Im Data Path findet die Verarbeitung der Daten statt.

Das Kernstück ist die Arithmetik Einheit (Arithmetik Logic Unit, ALU).

Die Register RØ bis R7 (General Purpose Register, GPR) stehen dem Benutzer zur Verfügung und haben zusammen mit weiteren 8 internen Registern (R10 bis R17) Zugang zur ALU. Die ALU greift, entsprechend dem auszuführenden Befehl, auf diese Register zu und verwendet sie als Rechenregister (AKKU), Datenzwischenspeicher oder Adressregister.

Register R7 wird als Programmzähler für die Maschine verwendet (auch PC=Programm Counter genannt). Der PC beinhaltet die Adresse des nächsten auszuführenden Befehls. Es handelt sich bei R7 also auch um ein generelles Register, das jedoch nur für Adressierung, nicht aber als Akkumulator oder für sonstige Operationen verwendet wird.

Das Register R6 wird normalerweise als Stack-Pointer verwendet und zeigt auf den letzten Eintrag im zugehörigen Stack (siehe dazu Kapitel 4.3).

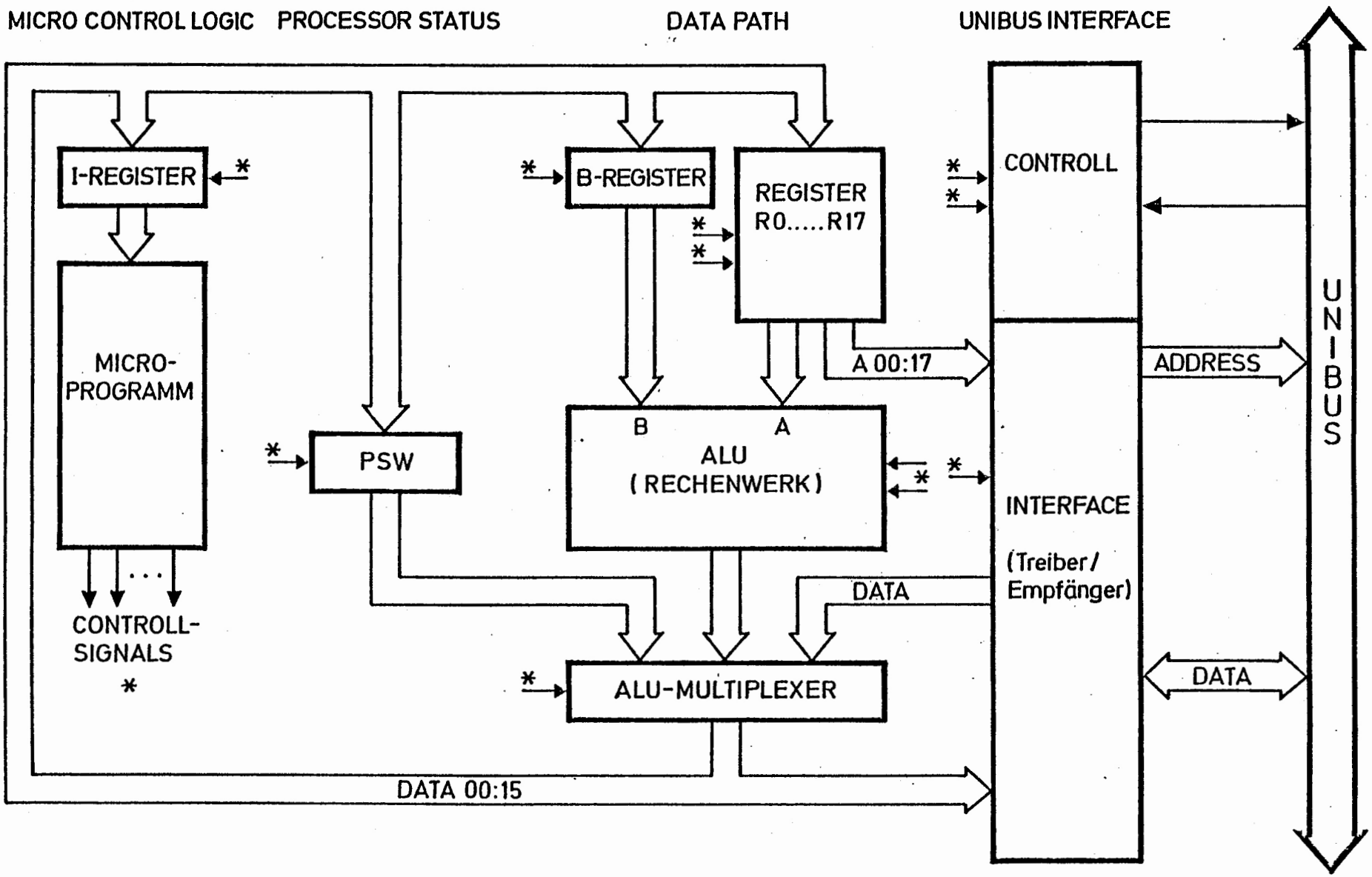

MICRO CONTROL LOGIC
PROCESSOR STATUS
DATA PATH
UNIBUS INTERFACE
I-REGISTER
MICRO-
PROGRAMM
CONTROLL-
SIGNALS
PSW
B-REGISTER
REGISTER
R0.....R17
A 00:17
B
A
ALU
(RECHENWERK)
ALU-MULTIPLEXER
DATA 00:15
CONTROLL
INTERFACE
(Treiber/
Empfänger)
DATA
ADDRESS
DATA
UNIBUS

Bild 2.2

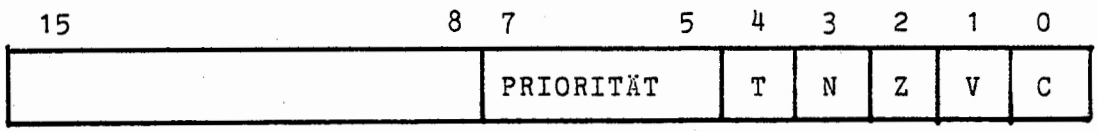
3. Micro Control Logic (Steuerlogik)

Diese Logik besteht im wesentlichen aus einem Instruktionsregister (I-Register, IR) welches den jeweils aktuellen Befehl beinhaltet und dem Instruktionsdecoder (Micro Control Unit) der die zur Ausführung des Befehls notwendigen Steuersignale erzeugt. Diese Einheit stellt praktisch einen Mikroprozessor dar, welcher zur Abarbeitung eines Befehls ein Mikroprogramm startet.

4. Prozessor - Statuswort (PSW)

Das PSW ist ein CPU-internes Register welches mit der Adresse 777 776 ansprechbar ist (bei LSI 11/03 nur mit 2 speziellen Befehlen).

Es beinhaltet Informationen über den Zustand des gerade laufenden Programms, im besonderen über die Art des Ergebnisses des zuletzt ausgeführten Befehls.



777 776

Priorität	Die Bits 5...7 geben die Priorität des gerade laufenden Programms an (Software-Priorität). Sie kann vom Programmierer frei zwischen Ø...7 gewählt werden. Sie ist zu unterscheiden von den Hardware-Prioritäten (vertikal, horizontal) welche einmal festgelegt und i. a. beibehalten werden (siehe Kap. 4.3.3)
T	Bit 4, Trace Trap, kann per Programm gesetzt werden und erzwingt nach jedem Befehl einen Trap nach 14
N	Bit 3, Negativ, wird automatisch gesetzt wenn das Ergebnis des letzten Befehls negativ war
Z	Bit 2, Zero, wird automatisch gesetzt wenn das Ergebnis des letzten Befehls null war
V	Bit 1, Overflow, wird automatisch gesetzt bei arithmetischem Überlauf (siehe Zahlenkreis Kap. 1.5.3), ebenso bei Carry
C	Bit Ø, Carry, wird automatisch gesetzt bei physikalischem Überlauf von Bit 15 bzw. Bit 7 (siehe Kap. 4.2.2)

2.2.3 Befehlsarten und Befehlsformate

1. Befehlsarten

Die Instruktionen des PDP-11 Befehlssatzes werden in verschiedene Gruppen eingeteilt. Die einzelnen Befehlsarten unterscheiden sich durch ihren Aufbau.

PDP-11-Instruktionen sind in 4 Gruppen eingeteilt:

1. Single-Operand-Befehle

2. Double-Operand-Befehle

3. Verzweigungsbefehle
   - Branch
   - Jump
   - Jump to Subroutine
   - Trap

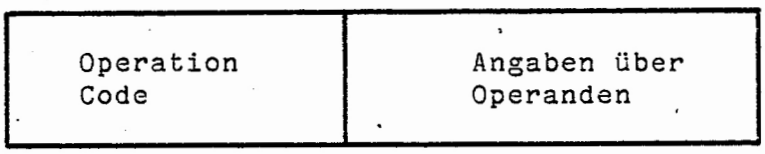
4. Sonstige
   - Condition-Code-Befehle
   - einige Spezial-Befehle

Gruppe 3 und 4 werden in Kaptel 4 besprochen.

2. Befehlsformate

Allgemeiner Befehlsaufbau

Da es nicht möglich ist in einem Wort einen Befehl und die dazugehörigen Operanden unterzubringen, wird der Befehl in einen Operationscode und Angaben darüber, wo die Operanden zu finden sind, aufgeteilt.



Instruktionen bestehen also aus:

1. **Op-Code**

   Maschinencode für den Befehl

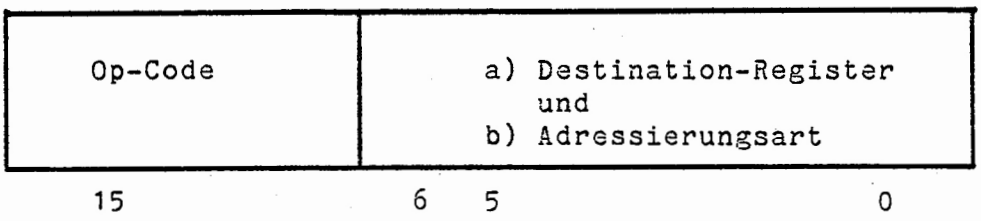
2. **Angaben über die Operanden**

   Hier wird angegeben
   a) Eines der Register RØ...R7
   b) Angabe wie das Register interpretiert werden soll (=Adressierungsart, Adressierungs-Mode)

Anmerkung: Die Adressierungsarten werden in Kapitel 3 ausführlich behandelt

Aufbau von Single-Operand-Befehlen

Diese Befehle beeinflussen nur einen Operanden, Destination (Zieloperand).



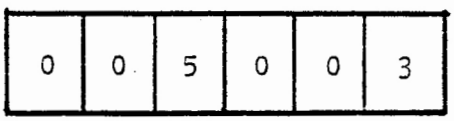
Ein Single-Operand-Befehl besteht aus:

1. Op-Code-Bits 6...15

2. a) Destination-Register Bits Ø...2
   (CPU-Reg. RØ...R7)
   b) Adressierungsart Bits 3...5

Beispiel: CLR R3 (Lösche das Register 3)

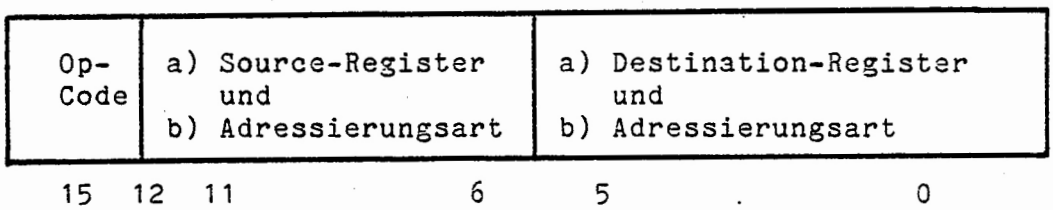
a) R3
b) Adressierungsart



Op-Code für
CLR

Aufbau von Double-Operand Befehlen

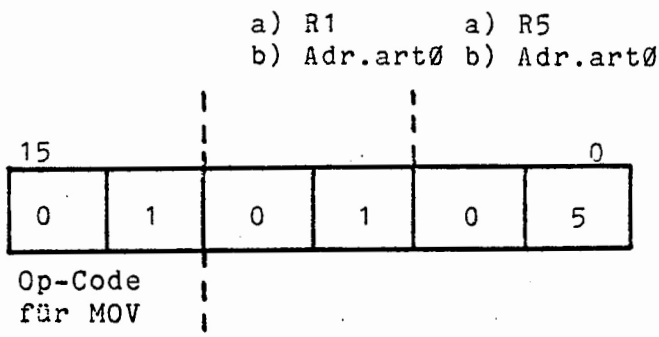
Diese Befehle beeinflussen zwei Operanden, Source (Quelloperand) und Destination (Zieloperand).



Ein Double-Operand-Befehl besteht aus:

1. Op-Code-Bits 12-15
2. a) Source-Register Bits 6...8 (RØ...R7)
   b) Adressierungsart Bits 9...11
3. a) Destination-Register Bits Ø...2 (RØ...R7)
   b) Adressierungsart Bits 3...5

Beispiel: MOV R1, R5 (Übertrage den Inhalt des Registers 1 ins Register 5)



(oktale Darstellung)

2.2.4 Arbeitszustände (Major States)

Der Prozessor kennt 5 Arbeitszustände, genannt "Major States". Jede Instruktion durchläuft bei seiner Ausführung in der CPU einige (oder alle) dieser Zustände, Bild 2.3.

1. Fetch

Nach Abarbeitung der vorangegangenen Instruktion schaltet der Prozessor in den Fetch-Major State. Die neue Instruktion wird aus dem Arbeitsspeicher gelesen, in das Instruktionsregister geladen und dekodiert. Von hier wird die Zentraleinheit in einen der nachfolgenden Arbeitszustände versetzt. Vorher wird noch der PC (=Programm Counter) um 2 erhöht!

2. Source

Wurde vom Instruktions-Dekorder eine "Doppel-Operand-Instruktion" erkannt, so schaltet der Prozessor in den Source-Major State. Der Source-Operand wird ermittelt. (z. B. Holen des Source-Operanden aus dem Speicher)

3. Destination

Nach Ausführung des Source-States, bzw. bei Single-Operand-Instruktionen nach dem Fetch-State geht der Prozessor in den Destination-State. Der Destination-Operand wird ermittelt.

4. Execute

Sind der Destination-Operand bzw. Source- und Destination-Operand ermittelt (d. h. sie befinden sich in der CPU), kann der Befehl ausgeführt werden.

5. Service

Bevor der nächste Befehl aus dem Speicher geholt wird, schaltet die CPU in den Service-Major-State. Hier wird das Prozessorstatuswort "aktualisiert" (vom Ergebnis des Befehls abhängig), eventuell vorliegende Interruptwünsche und Traps können jetzt (und nur jetzt) bedient werden, und falls die Halt-Taste gedrückt wurde so bleibt die CPU jetzt stehen.

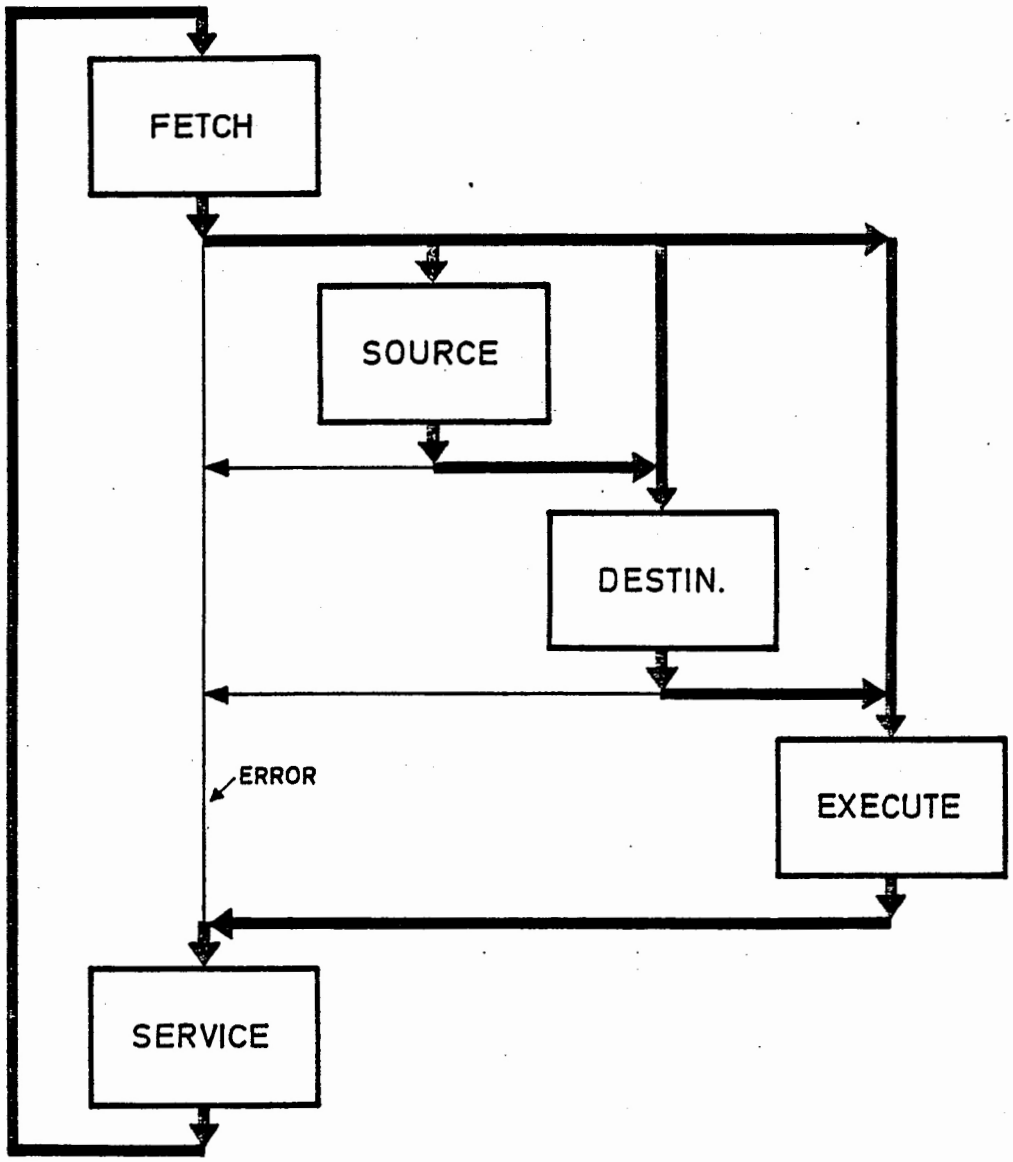
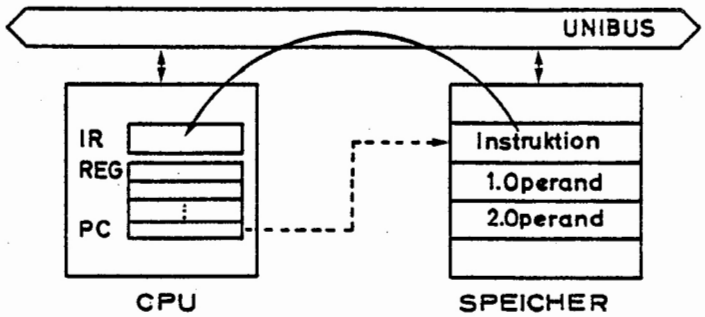
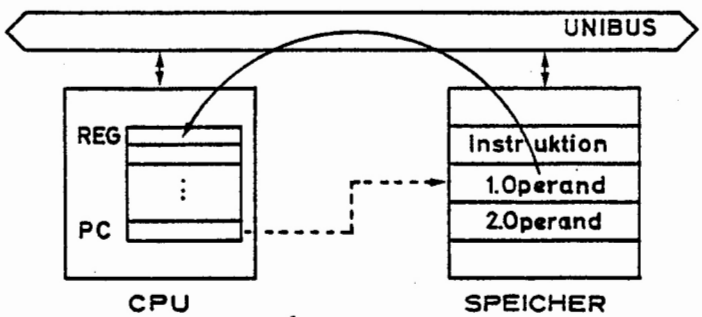
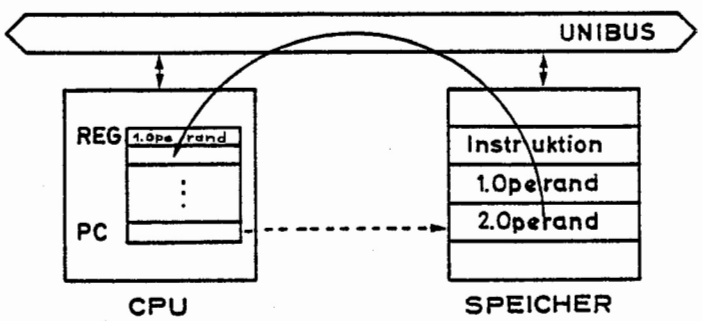
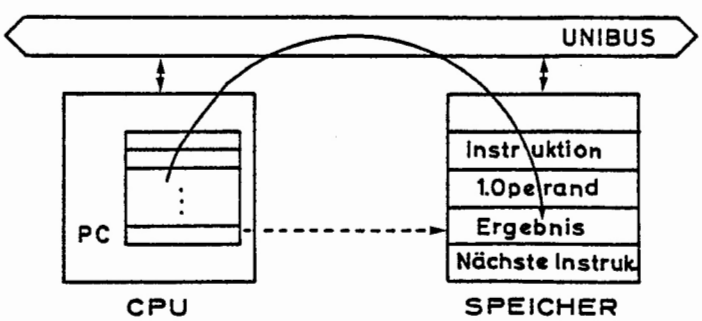


Bild 2.3 Arbeitszustände (Major States)

Beispiel

Ausführung eines Doppel-Operand-Befehls, wobei beide Operanden und das Ergebnis im Speicher stehen sollen (z. B. ADD A,B)

VORGANG	BESCHREIBUNG	DIAGRAMM
FETCH	Holen der Instruktion auf die der PC zeigt, in die CPU (IR); erhöhen des PC um 2; dekodieren	
SOURCE	Holen des 1. Operanden (Source-Operand) in die CPU (int.Reg.); erhöhen des PC um 2.	
DESTI-NATION	Holen des 2. Operanden (Destination-Operand) in die CPU (int.Reg.)	
EXE-CUTE	Ausführung der Instruktion gemäß Befehls code; Abspeichern des Ergebnisses.	
SERVICE	Prozessor-Statuswort setzen; Abfrage nach Interrupt, Trap oder Halt-Taste und ggf. Sprung ins Bedienungsprogramm. "Normalfall": Holen der nächsten Instruktion (Fetch)	

2.3 Adress- und Speicherbereich

Die Kapazität eines Speichers wird in xK-Worten oder in xK-Bytes angegeben (1 K ≙ Faktor 1024). Da man in der PDP 11 sowohl mit Wort-, als auch mit Byte- Adressierung arbeitet, muß jedem Byte ein Speicherplatz zugeordnet werden. Durch die Anordung: High- und Lowbyte in einem Wort ergeben sich gerade Adressen für Worte und Lowbytes, sowie ungerade Adressen für Highbytes.

2.3.1 Adressbereiche

16-Bit Adressen:

Mit 16-Bit-Adressen können 32K Worte adressiert werden (64k Bytes) Davon sind 28K für die Programmierung zugänglich (Arbeitsspeicherbereich) und die restlichen, obersten 4K sind reserviert für I/O-Deviceregister und CPU-Register, wie PSW und GPR's. (I/O-Page)

Mit 16 Bit können die oktalen Zahlen von Ø bis $177\ 777_8$ gebildet werden d. h.: $200\ 000_8$ versch. Adressen sind möglich Dezimal ausgedrückt : $65\ 536_{10}$ oder 64K Adressen

18-Bit Adressen:

Mit 18-Bit-Adressen, die man unter Verwendung einer "Memory-Management"-Einrichtung erzeugen kann, ist es möglich 128K Worte (256K Bytes) zu adressieren.

Mit 18 Bit können die oktalen Zahlen von Ø bis $777\ 777_8$ gebildet werden d. h.: $1\ 000\ 000_8$ versch. Adressen sind möglich. Dezimal sind dies 262 144 bzw. 256K Adressen.

22-Bit-Adressen (nur 11/44 und 11/70)

Hier erreicht man einen Adressbereich von 2048K Worten

2.3.2 Register- und Geräteadressen (I/O-Page)

Alle Adressen ab 160000 werden nicht zur Adressierung des Arbeitsspeichers verwendet. Sie sind reserviert für die Register der CPU und für externe Geräte bzw. deren Interface-Module (I/O-Page)

Da der UNIBUS über 18 Adr.Leitungen verfügt, die CPU aber nur 16 Bit Adressen verarbeiten kann (Ausnahme: Memory Management), werden die Bits 16 und 17 der Adresse per Hardware wie folgt erzeugt:

Bit	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø
	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X

X = Ø oder 1

Das bedeutet:
Bei allen Adressen von $160\ 000_8$ bis $177\ 777_8$ (≙ I/O Page) ist das UND-Gatter erfüllt (alle Eing. ="1") und die tatsächlich (physikalisch) auf dem Adressbus entstehenden Adressen der I/O-Page liegen zwischen $760\ 000_8$ und $777\ 777_8$

Beispiel: a) In einem Programm wird die Adresse 1000 (=virtuelle Adresse) verwendet. Auf dem Adressbus ensteht ebenfalls 1000 (=phys. Adresse), da das UND-Gatter nicht erfüllt ist. Mit dieser Adresse wird eine Arbeitsspeicherzelle angesprochen.

b) In einem Programm wird die Adresse 177 566 (=virtuelle Adresse) verwendet, das UND-Gatter ist erfüllt d. h. es entsteht auf dem Adressbus 777 566 (=Phys.Adresse). Damit wird der Printer-Buffer des Consol-Interface angesprochen.

2.3.3 Speicherbereich und I/O-Page

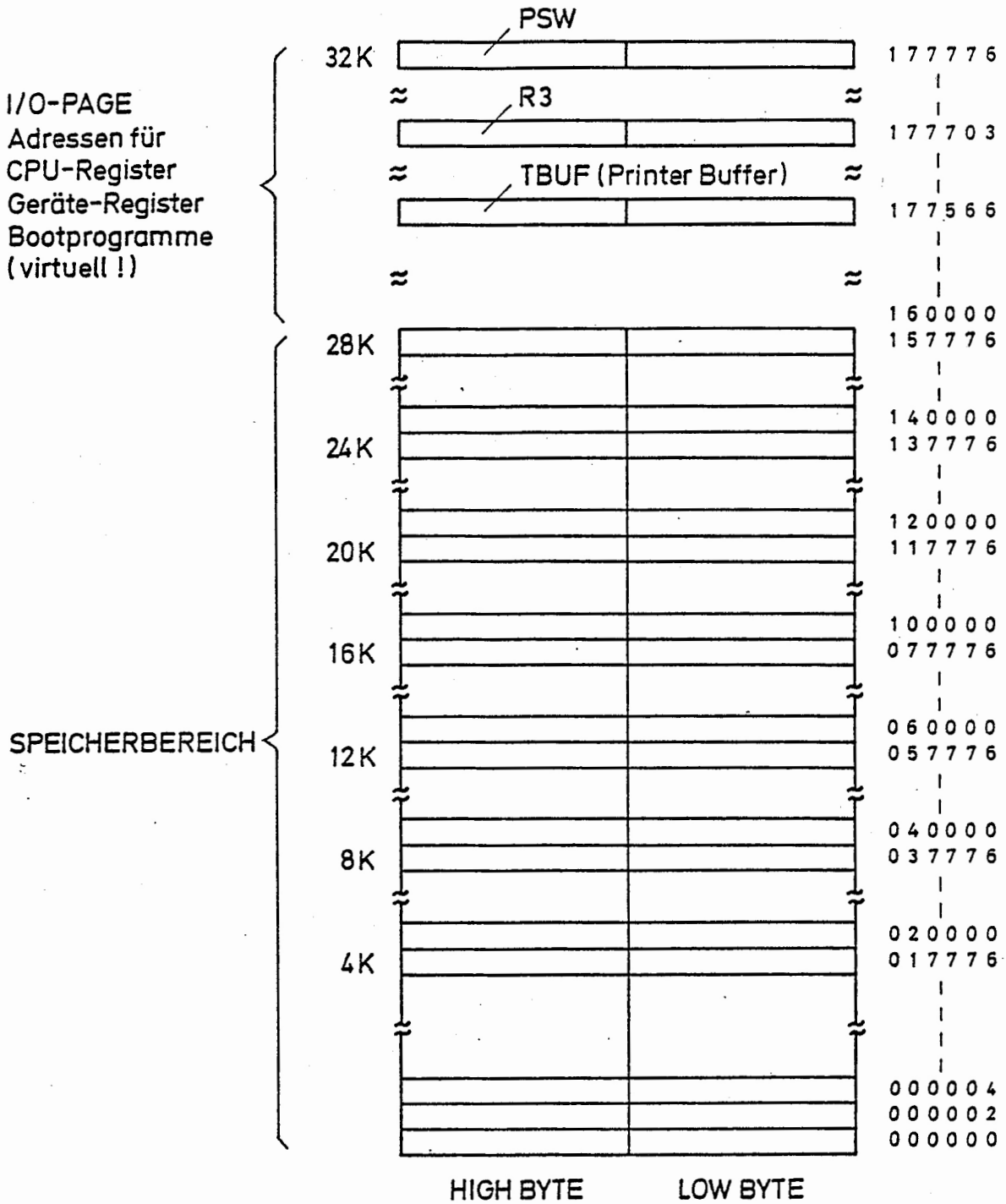


Bild 2.4

Anmerkung zu Bild 2.4

1. Die angegebenen (gradzahligen) Adressen beziehen sich bei Wortzugriff auf das ganze Wort, bei Bytezugriff auf das Low-Byte.
   Bei Bytezugriff auf das High-Byte muß die entsprechende ungradzahlige Adresse angegeben werden.

2. Wie in Kap. 2.3.2 dargestellt, liegt der virtuelle Bereich von 160 000 bis 177 776 physikalisch bei 776 000 bis 777 776. Im Bereich Ø bis 157 77 sind die virtuellen Adressen identisch zu den physikalischen (ohne Memory Management!)

3. Gedankenstütze:
   4K Worte entsprechen $20\ 000_8$ Adressen
   16K Worte entsprechen $100\ 000_8$ Adressen

2.3.4 Speicherorganisation (Memory Mapping)

Einige Bereiche im Arbeitsspeicher werden nicht für die freie Programmierung verwendet, sondern sind für spezielle Zwecke reserviert. (Bild 2.5)

1. Bereich der TRAP-Vektoren

Verschiedene Fehler im Programm oder im System bewirken einen TRAP. Dies ist eine (Programm-) Unterbrechung die zur Folge hat, daß automatisch eine (für jeden Fehler definierte) Adresse im Bereich zwischen Ø u. 400_8 auf den Adressbus ausgegeben wird (=Vektor). Unter diesem Vektor findet der Prozessor eine Startadresse für eine Fehlerroutine.

2. Bereich der INTERRUPT - Vektoren

Der obengenannte Adressbereich wird ebenfalls automatisch angewählt, wenn ein Peripheriegerät eine Unterbrechung des Hauptprogrammes verursacht. Das Gerät sendet selbst seinen Interrupt-Vektor mit dessen Hilfe der Prozessor die Startadresse der Interruptroutine findet.

3. Bereich des Hardware-STACK

Der Bereich von Adresse 400_8 (STACKLIMIT) bis zur niedrigsten vom Programm benutzten Adresse (z. B. 1000_8) kann als STACK benutzt werden. Die Benutzung dieses Bereichs wird in den Kapiteln über Interrupt- und Unterprogrammierung erläutert (ab Kap. 4.3).

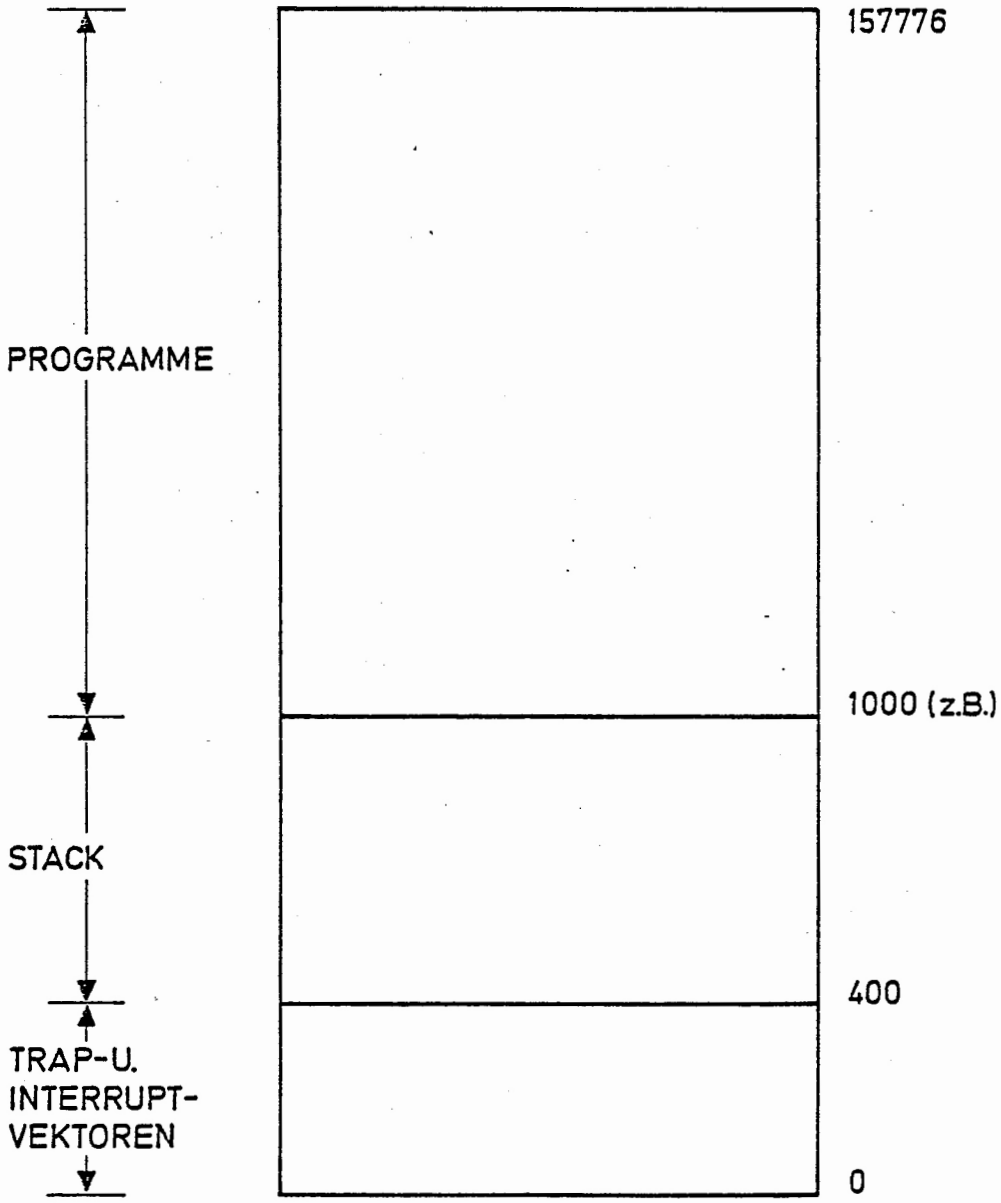

157776
PROGRAMME
1000 (z.B.)
STACK
400
TRAP-U.
INTERRUPT-
VEKTOREN
0

Bild 2.5

Arbeitsblatt 2.1

1. Nach welchem Prinzip erfolgt die Kommunikation auf dem UNIBUS?

2. Kann der Arbeitsspeicher den UNIBUS anfordern?

3. Nennen Sie die 4 Hauptbestandteile der CPU!

4. Nennen Sie die 4 Gruppen der PDP11-Befehle!

5. Skizzieren Sie den Aufbau eines Double-Operand-Befehls!

6. Wie heißen die Arbeitszustände der CPU?

7. a) Wie heißen die obersten 4K des Adressbereiches?
   b) Können in diesem Bereich Arbeitsspeicherzellen liegen?
   c) Bei welcher Adresse (physikalisch und virtuell) beginnt dieser Bereich?

8. In welchem Bereich liegen die Vektoradressen und der STACK?

9. Wie groß ist die kleinste adressierbare Einheit im Arbeitsspeicher?

KAPITEL 3 - ADRESSIERUNGSARTEN

Arbeitsblätter 3.1, 3.2

3.1. Einführung

Als Einführung in das Kapitel ADRESSIERUNGSARTEN soll hier zunächst einiges aus früheren Kapiteln zusammengefaßt werden:

Die Daten im Arbeitsspeicher werden mit Hilfe von Befehlen (Instruktionen) verarbeitet.

Eine PDP 11 - Instruktion enthält gewöhnlich:

1. Die Art der Verarbeitung (ADD, MOV usw.), also den Operationscode.

2. Das CPU-Register, welches den Source-Operanden oder dessen Adresse, und / oder das CPU-Register, welches den Destination-Operanden oder dessen Adresse enthält.

3. Die Adressierungsart, d.h. wie die angegebenen Register zu interpretieren sind.

Abhängig von der Adressierungsart können die CPU-Register wie folgt verwendet werden:

1. Als Akkumulator, um die Daten direkt im Register zu verarbeiten, d.h. der Inhalt des Registers ist der Operand selbst.

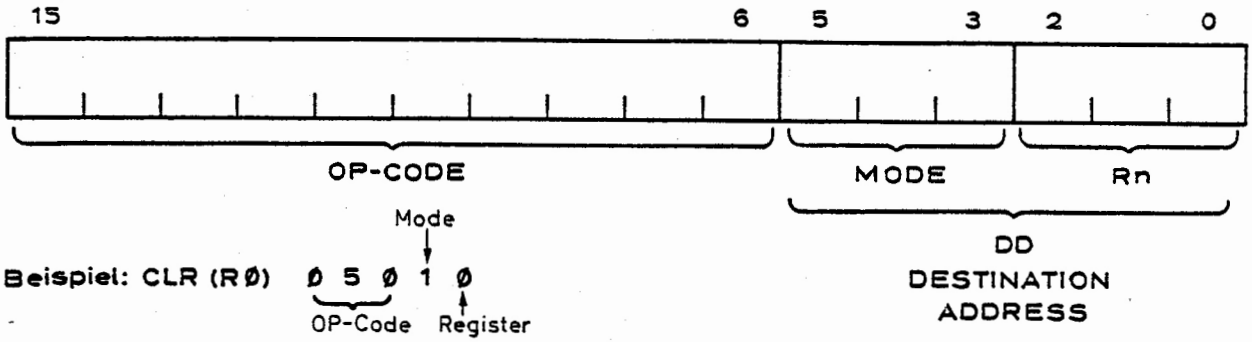
2. Als Zeiger auf einen Operanden, d.h. der Inhalt des Registers ist die Adresse des Operanden.

3. Als automatischer Zeiger, der nach jeder Benutzung automatisch erhöht oder erniedrigt wird.

4. Als Indexregister, wobei der Inhalt des Reg. und das nächste, auf den Befehl folgende Wort addiert werden. Die Summe stellt die Adresse des Operanden dar. Dies wird hauptsächlich für variable Listen verwendet.

Aufbau von Single- und Double-Operand-Befehlen

SINGLE OPERAND-BEFEHL

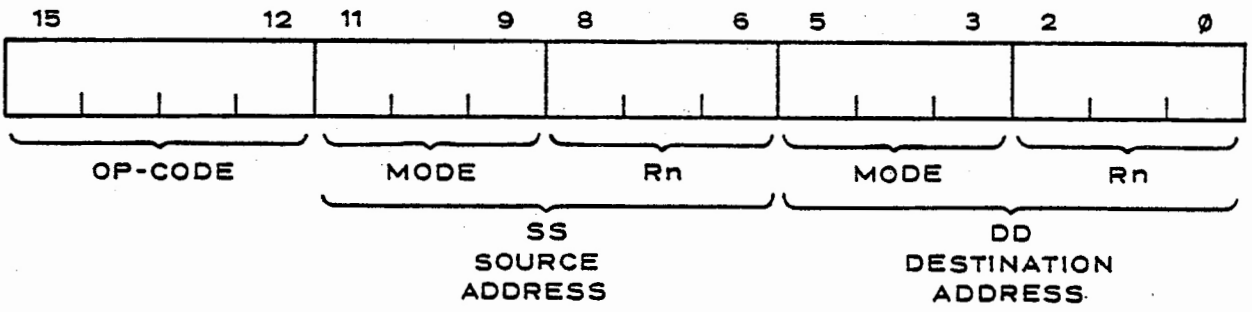


Beispiel: CLR (RØ) Ø 5 Ø 1 Ø

Mode

OP-Code Register

DOUBLE OPERAND-BEFEHL



Beispiel: MOV R1,R2 Ø 1 Ø 1 Ø 2

Mode Mode

OP-Code Register Register

Die Adressierungsarten (Adressing Modes) werden in 2 Gruppen eingeteilt:

1. Basic Modes: verwendet werden die Register RØ...R6

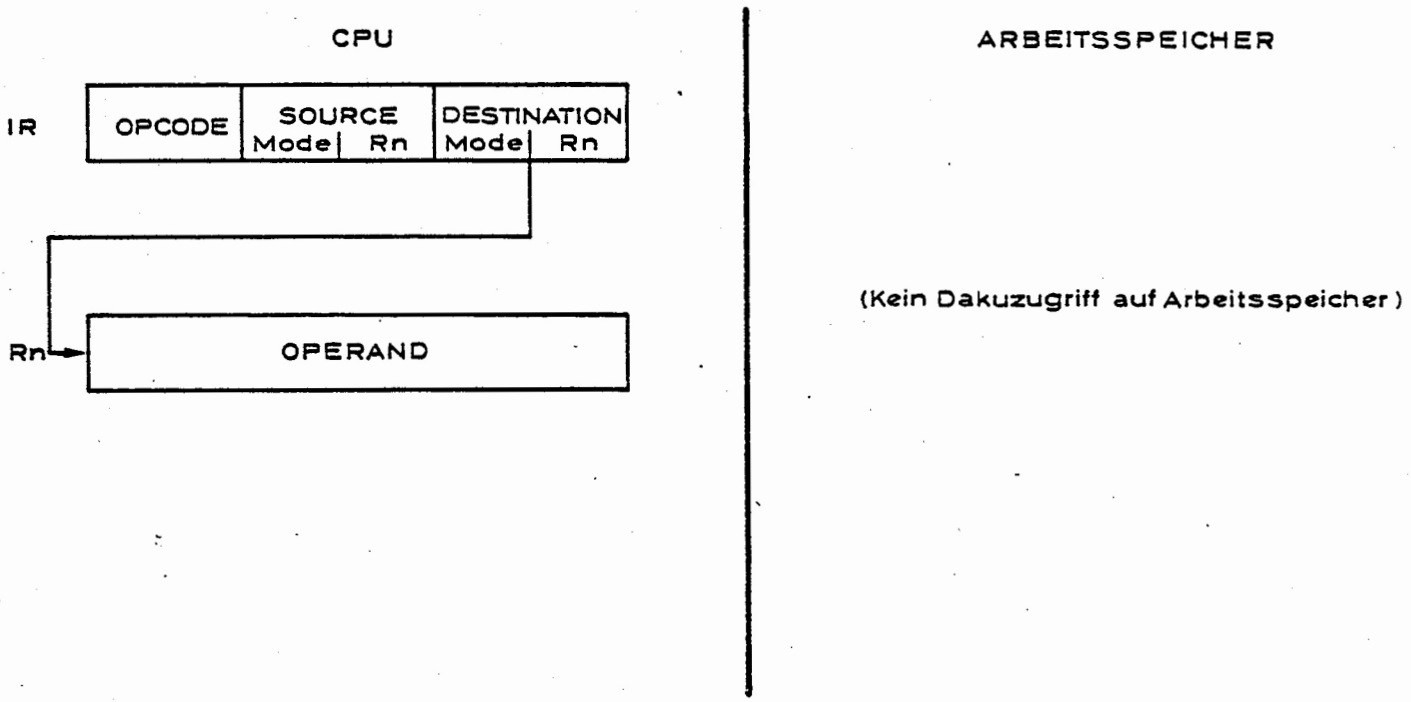
2. PC Modes: verwendet wird das Register R7 (=PC)

3.2 Basic Modes

3.2.1 Mode 0, Register Mode

Der Inhalt des Registers ist der Operand. Dieser Mode ermöglicht schellsten Zugriff zu den Daten, da der Weg über den Unibus entfällt. (IR=Instruktionsregister)

Symbol: Rn oder %n n = Ø...7



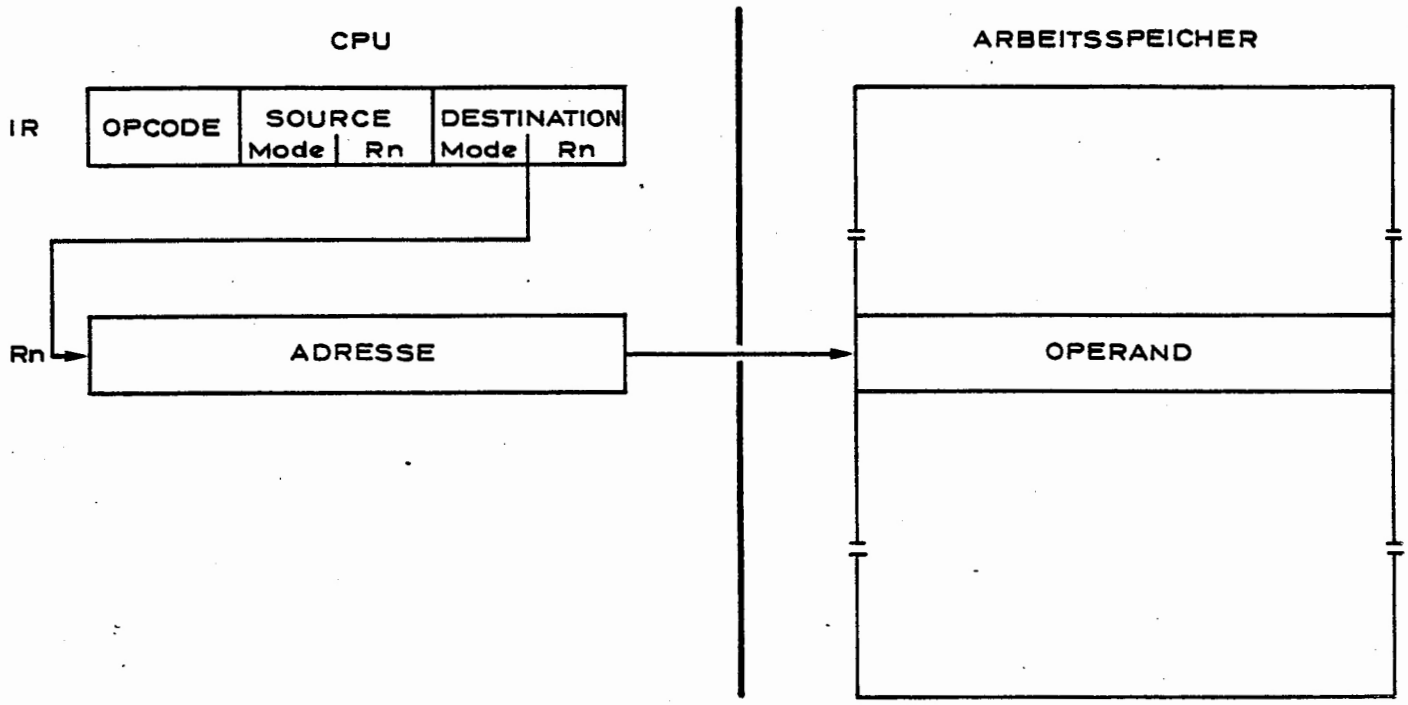
Beispiele für Mode Ø

symbolisch	oktaler Code	Bedeutung
1. INC RØ	005200	Erhöhe Register RØ um 1
2. ADD R1,R2	060102	Addiere R1 zu R2

3.2.2 Mode 1, Register Deferred Mode

Das Register enthält die Adresse eines Operanden, der entweder im Arbeitsspeicher oder in einem I/O-Register zu finden ist. Anders ausgedrückt: Der Inhalt des Registers zeigt auf den Operanden.

Symbol: (Rn) n = Ø...6



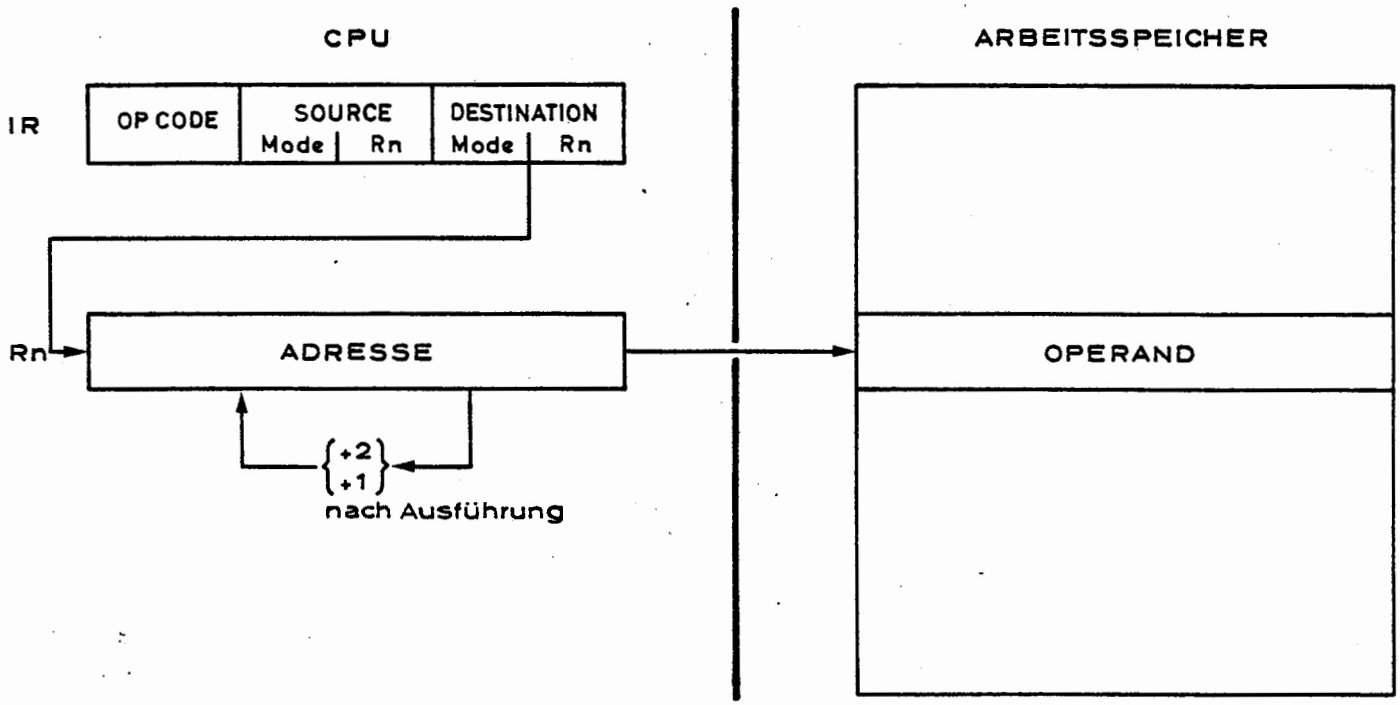
Beispiele für Mode 1

symbolisch	oktaler Code	Bedeutung
1. MOV R1,(R4)	010114	Lade Inhalt von R1 in die Speicherzelle auf die R4 zeigt
2. MOV (R1),(R4)	011114	Lade Inhalt der Speicherzelle auf die R1 zeigt in die Speicherzelle auf die R4 zeigt

3.2.3 Mode 2, Autoincrement Mode

Der Inhalt des Registers ist die Adresse eines Operanden. Diese Adresse wird nach Ausführung der Operation um 1 bei Byte- und um 2 bei Wort-Instruktionen erhöht.
Dieser Mode ermöglicht das automatische Abarbeiten von Listen.

Symbol: (Rn)+ n = Ø...6



Beispiele für Mode 2

1. Beispiel

symbolisch	oktaler Code	Bedeutung
CLR (R5)+	005025	Lösche das Speicherwort auf das R5 zeigt. Erhöhe R5 anschließend um 2

VOR DER AUSFÜHRUNG

20000 005025	Register
30000 111116	R5 030000

NACH DER AUSFÜHRUNG

20000 005025	Register
30000 000000	R5 030002

2. Beispiel

symbolisch	oktaler Code	Bedeutung
CLRB (R5)+	105025	Lösche das Speicherbyte auf das R5 zeigt. Erhöhe R5 anschließend um 1

VOR DER AUSFÜHRUNG

20000 105025	Register
30000 111116	R5 030000

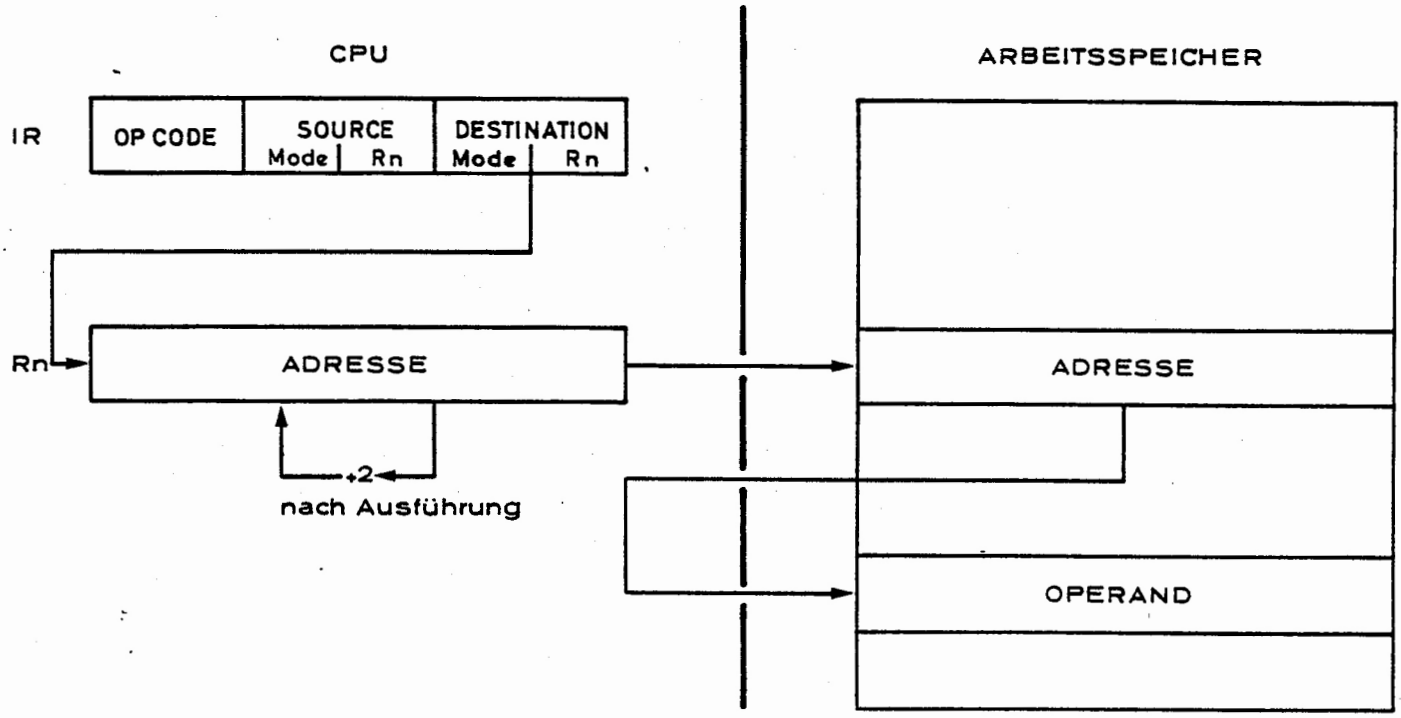
NACH DER AUSFÜHRUNG

20000 105025	Register
30000 111000	R5 030001

3.2.4 Mode 3, Autoincrement Deferred Mode

In dieser Adressierungsart wird der Inhalt des Registers als Zeiger benutzt, der zu einer Adresse eines Operanden zeigt.
Nachdem der Befehl abgearbeitet ist, wird der Inhalt des Registers automatisch um 2 erhöht.

Symbol: @(Rn)+ n = Ø...6



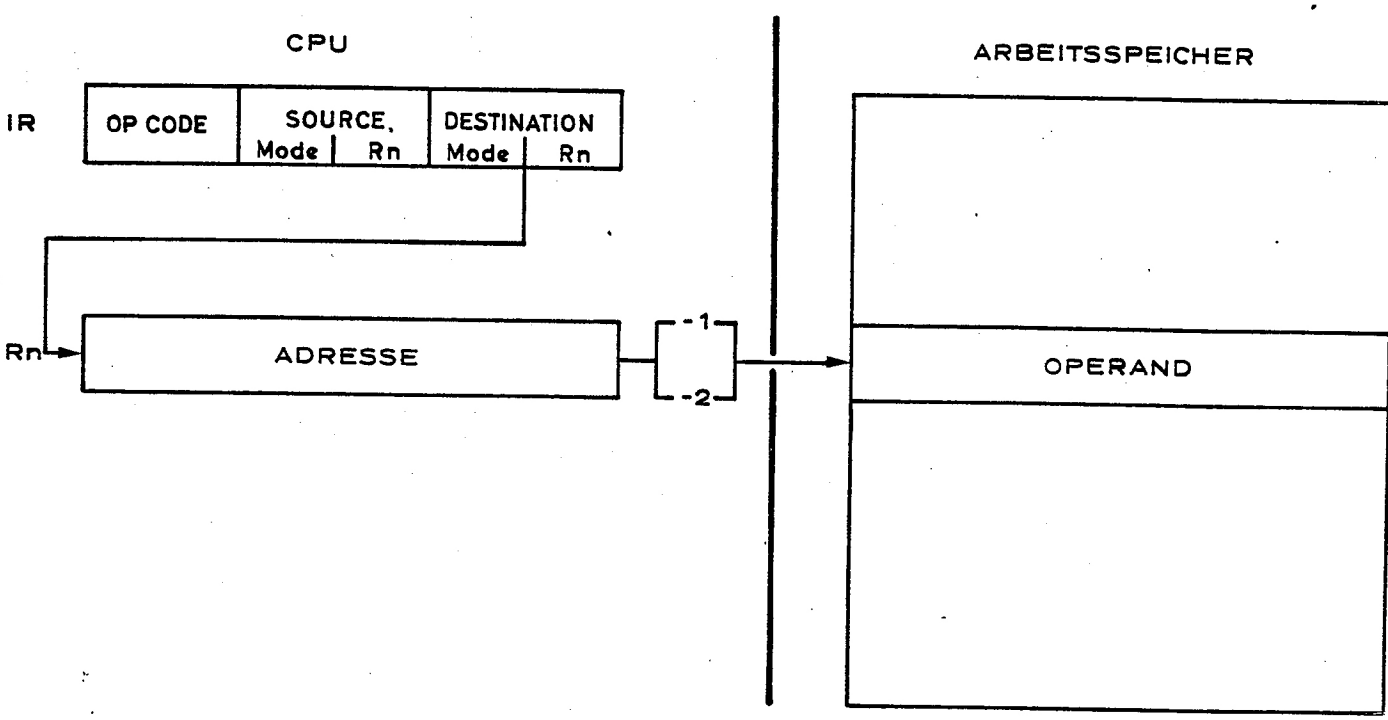
Beispiel für Mode 3

symbolisch	oktaler Code	Bedeutung
INC @(R2)+	005232	R2 zeigt auf eine Speicherzelle welche eine Adresse beinhaltet.Der Inhalt dieser Adresse soll um 1 erhöht werden. Anschließend R2 um 2 erhöhen

3.2.5 Mode 4, Autodecrement Mode

Der Inhalt des benutzten Registers ist die Adresse eines Operanden, welche vor Ausführung des Befehls je nach verwendetem Format um 1 bzw. um 2 erniedrigt wird.

Symbol: -(Rn) n = Ø...6



Beispiel für Mode 4

symbolisch	oktaler Code	Bedeutung
INC -(RØ)	005240	Der Inhalt von R2 wird um 2 dekrementiert und als Adresse des Operanden verwendet. Der Operand wird um 1 erhöht

VOR DER AUSFÜHRUNG

1000	005240	Register
7774	000000	R0 0017776

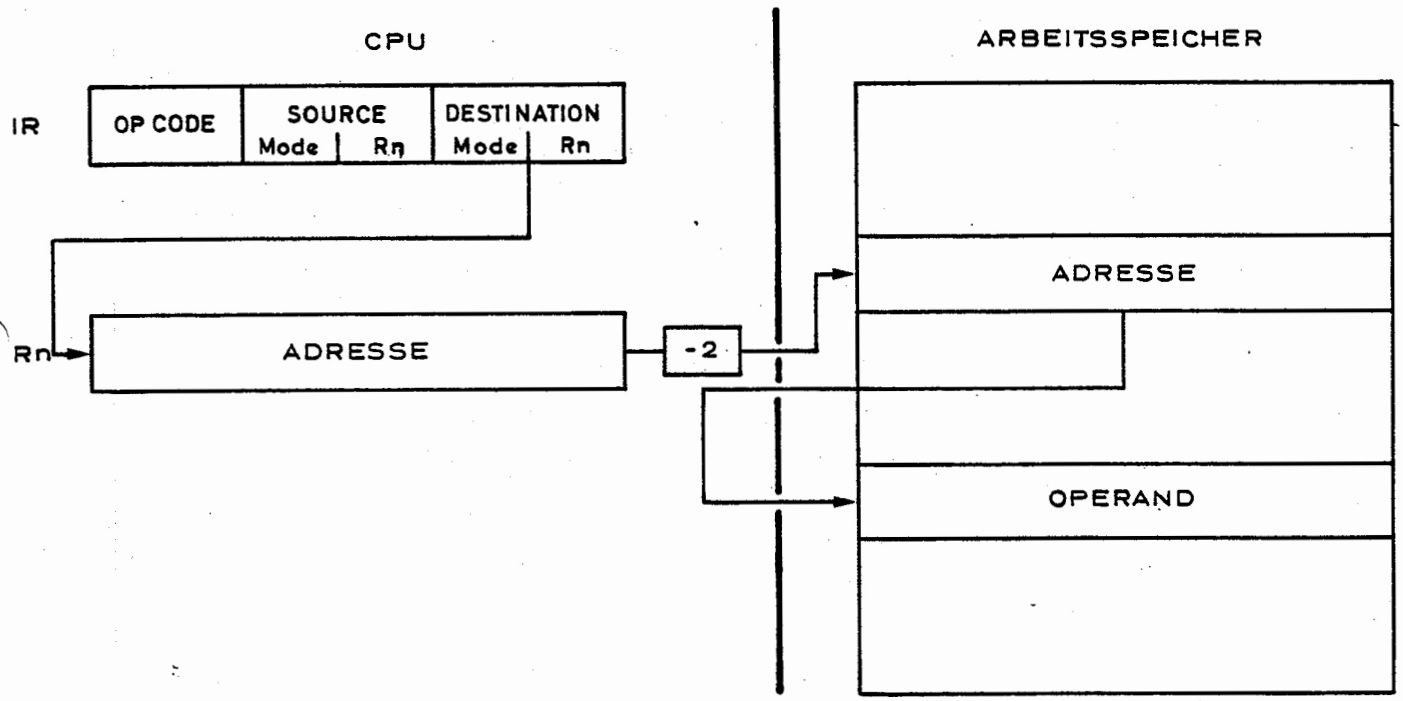
NACH DER AUSFÜHRUNG

1000	005240	Register
7774	000001	R0 0017774

3.2.6 Mode 5, Autodecrement Deferred Mode

Das Register enthält einen Zeiger der zuerst um 2 erniedrigt wird und anschließend auf die Adresse einer Adresse eines Operanden zeigt.

Symbol: @-(Rn) n= Ø...6



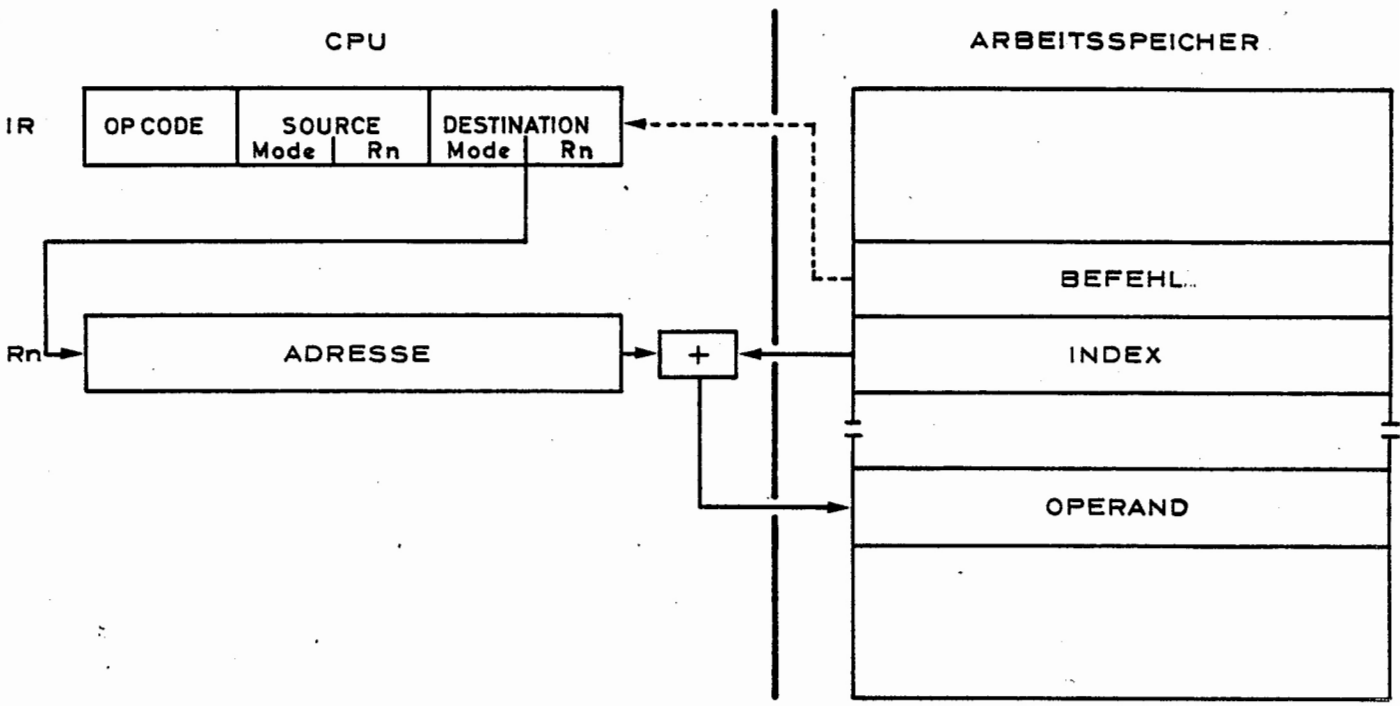
Beispiel für Mode 5

symbolisch	oktaler Code	Bedeutung
COM@-(RØ)	005150	Der Inhalt von RØ wird um 2 erniedrigt und als Adresse der Adresse des Operanden verwendet. Aus dem Operanden wird das Einer-Komplement gebildet.

3.2.7 Mode 6, Index Mode

In diesem Mode wird dem Befehl ein Index beigefügt, der zum Registerinhalt addiert wird.
Die Summe ist die effektive Adresse eines Operanden. Index und Registerinhalt bleiben jedoch unverändert!

Symbol: X(Rn) n= Ø...6



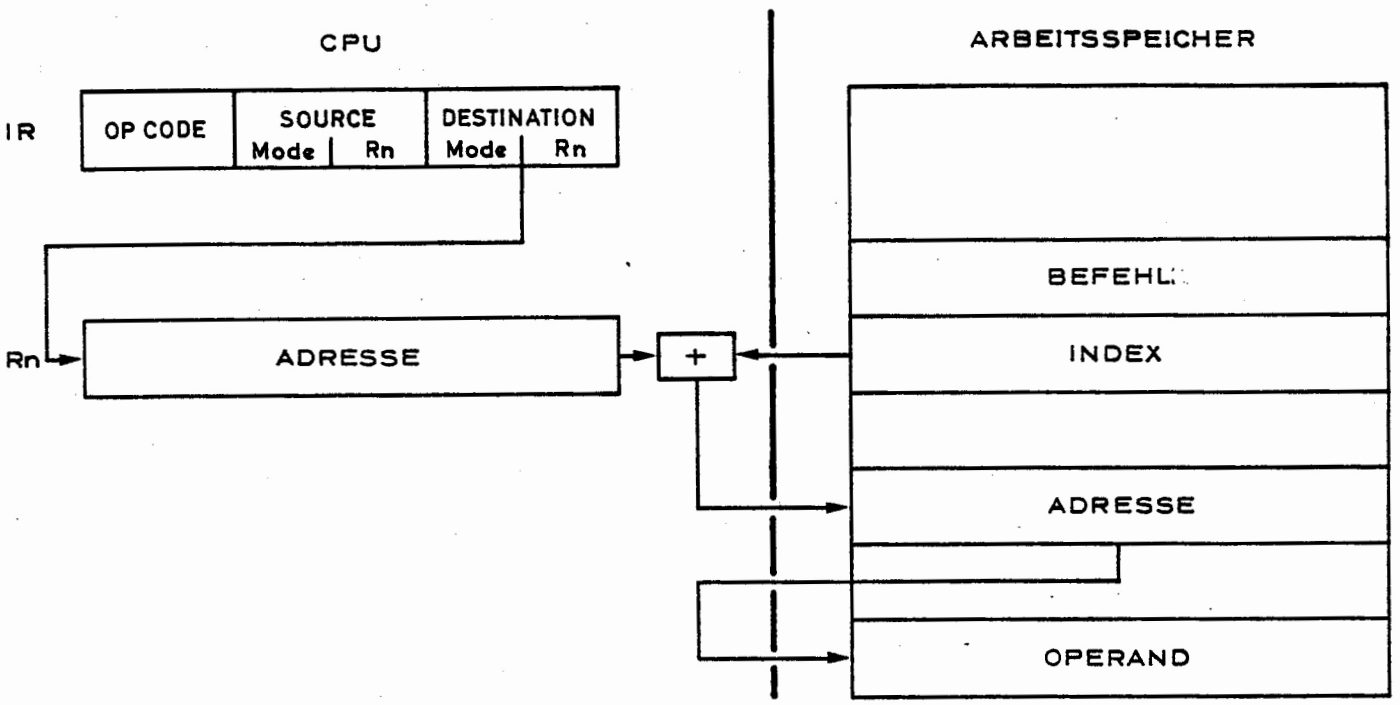
Beispiel für Mode 6

symbolisch	oktaler Code	Bedeutung
CLR 200(R4)	005064 000200	Die Adresse des Operanden wird durch die Addition von 200 zum Register R4 errechnet. Der Operand wird gelöscht.

3.2.8 Mode 7, Index Deferred Mode

Wie Mode 6, doch ist hier die Summe als Zeiger auf eine Adresse zu verstehen.

Symbol: @X(Rn) n= Ø...6



Beispiel für Mode 7

symbolisch	oktaler Code	Bedeutung
ADD @1000(R2),R1	067201 001000	1000 und der Inhalt von R2 summiert ergeben die Adresse der Adresse des Source-Operanden. Dieser Inhalt wird dann zu R1 addiert.

3.3 PC Modes

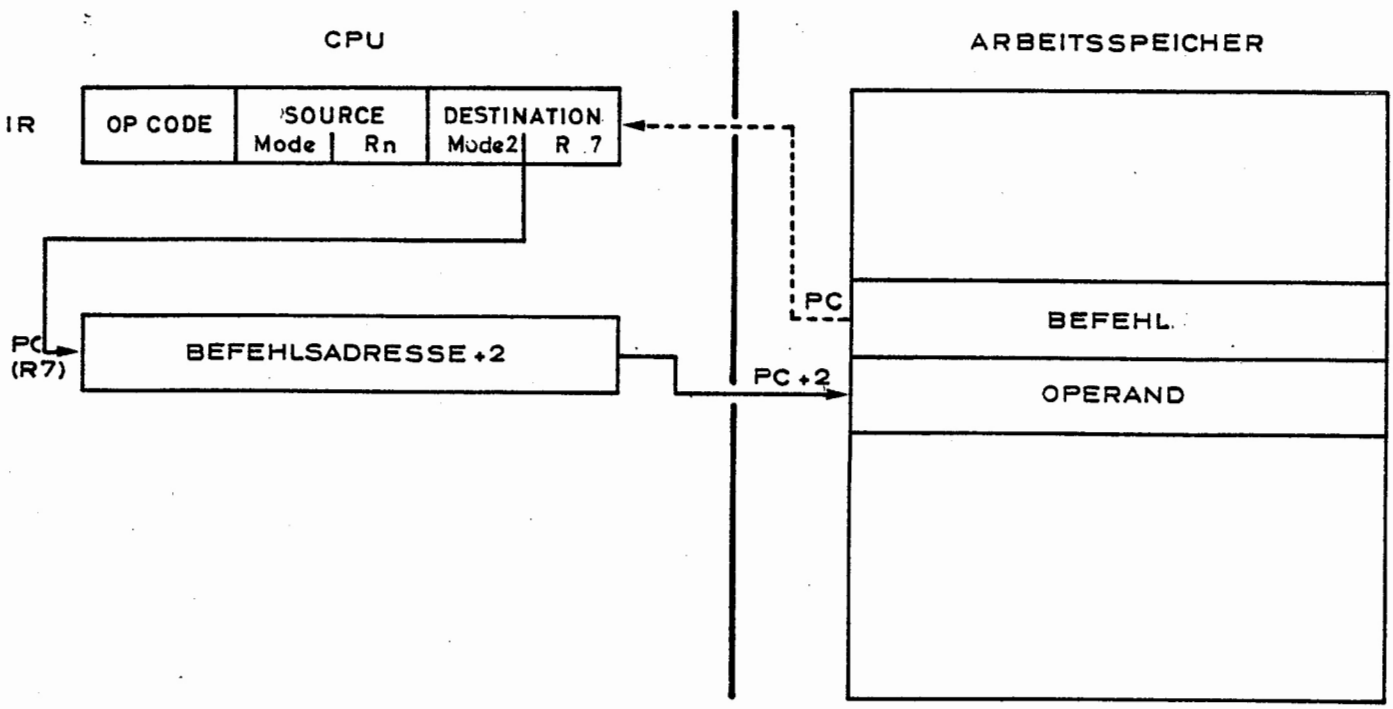
Diese Adressierungsarten benutzen als Register den Programmzähler Register 7.
Register 7 unterscheidet sich von den übrigen Registern (RØ...R6) dadurch, daß es nach dem Holen jeder Instruktion automatisch um 2 erhöht wird. Diese Eigenschaft wird in den folgenden Modes ausgenutzt.

Programme die mit PC Modes geschrieben werden, haben den besonderen Vorteil, daß sie an jeder Stelle des Arbeitsspeichers lauffähig abgelegt werden können. Der Programmierer braucht also vorher den ihm zur Verfügung stehenden Speicherbereich nicht zu kennen, wenn er mit PC Modes (und Branch-Befehlen, siehe später) programmiert. Diese Programme sind positions-unabhängig, da alle Verweise relativ zum aktuellen PC erzeugt werden (=position independent coding, PIC).

3.3.1 Mode 2/Register 7, Immediate Mode

Diese Adressierungsart ist ähnlich dem Autoincrement Mode, nur wird hier der auf die Instruktion folgende Wert als Operand interpretiert.

Symbol: # n



Beispiel zu Mode 2, Reg. 7

symbolisch	oktaler Code	Bedeutung
ADD # 10,RØ	062700 000010	Der Wert 10 steht unmittelbar hinter dem Befehl und wird zum Inhalt von R0 addiert. Die CPU holt den Befehl und erhöht den PC automatisch um 2. Damit zeigt der PC auf den Operanden. Dieser wird geholt und auf RØ addiert. Wegen Mode 2 wird nun der PC um 2 erhöht und zeigt auf den nächsten Befehl.

VOR DER AUSFÜHRUNG

1020	062700	Register
1022	000010	R0 000020
1024		PC 001020

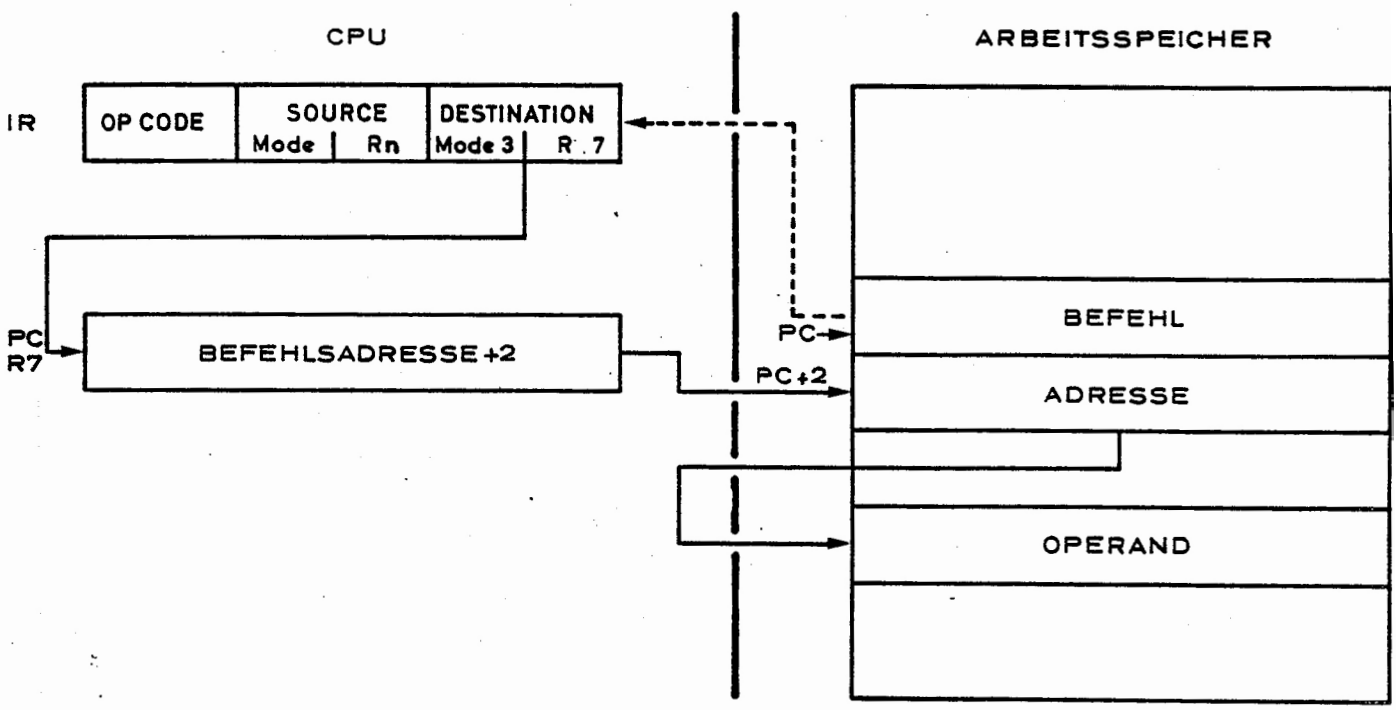
NACH DER AUSFÜHRUNG

1020	062700	Register
1022	000010	R0 000030
1024		PC 001024

3.3.2 Mode 3/Register 7, Absolute Mode

Hier wird der Autoincrement Deferred Mode mit Reg. 7 verwendet. Nach dem Holen des Befehls wird der PC automatisch um 2 erhöht. Der PC zeigt nun auf den Speicherplatz hinter dem Befehl, welcher die (absolute) Adresse eines Operanden enthält.

Symbol: @#A



Beispiel für Mode 3 Reg. 7

symbolisch	oktaler Code	Bedeutung
CLR @#1100	005037 001100	Lösche den Inhalt der Adresse 1100

VOR DER AUSFÜHRUNG

2000	005037	Register
2002	001100	PC 002000
1100	177777	

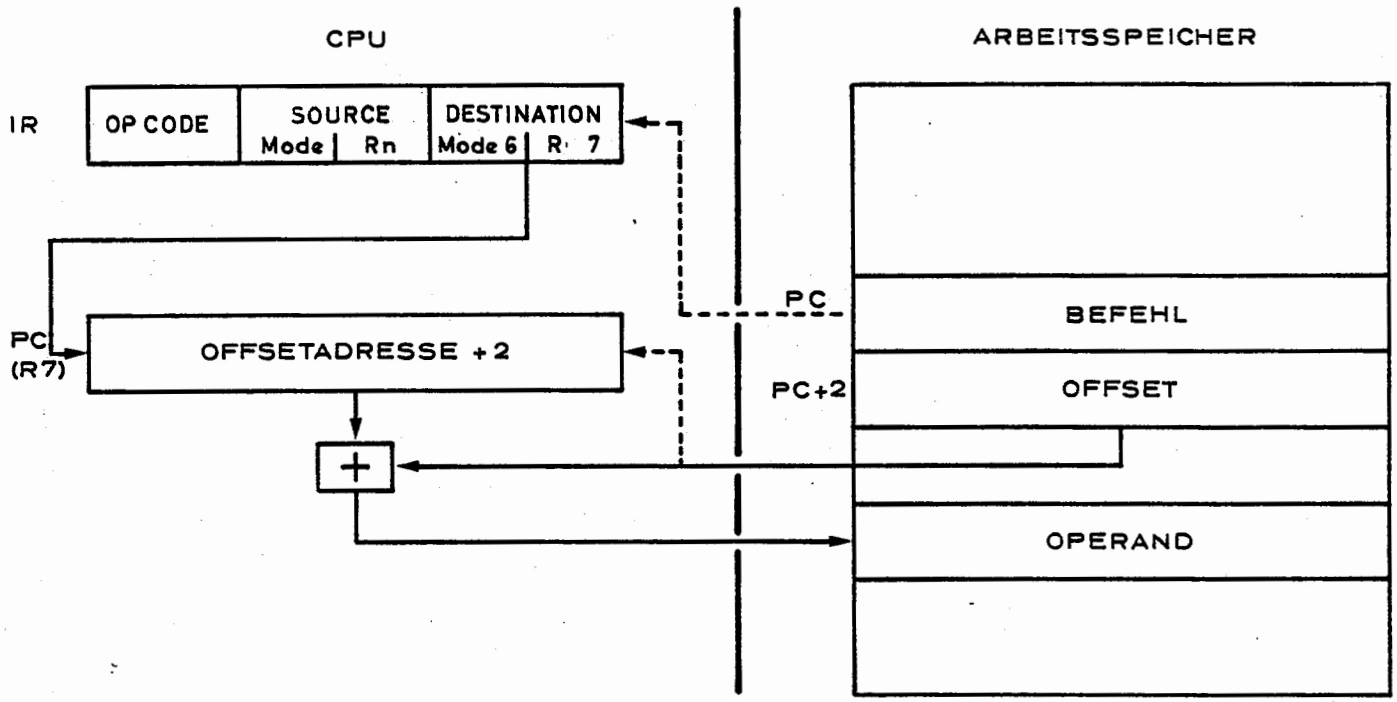
NACH DER AUSFÜHRUNG

2000	005037	Register
2002	001100	
2004		PC 002004
1100	000000	

3.3.3 Mode 6/Register 7, Relative Mode

Zuerst wird der PC um 2 erhöht und zeigt auf ein Offsetwort, welches zum nächstfolgenden PC-Wert addiert wird. Die entstandene Summe ist die Adresse "A" eines Operanden.

Symbol: A



Kurz:

```
    Offsetadresse
+   Offset
+   2
=   Adresse des Operanden
```

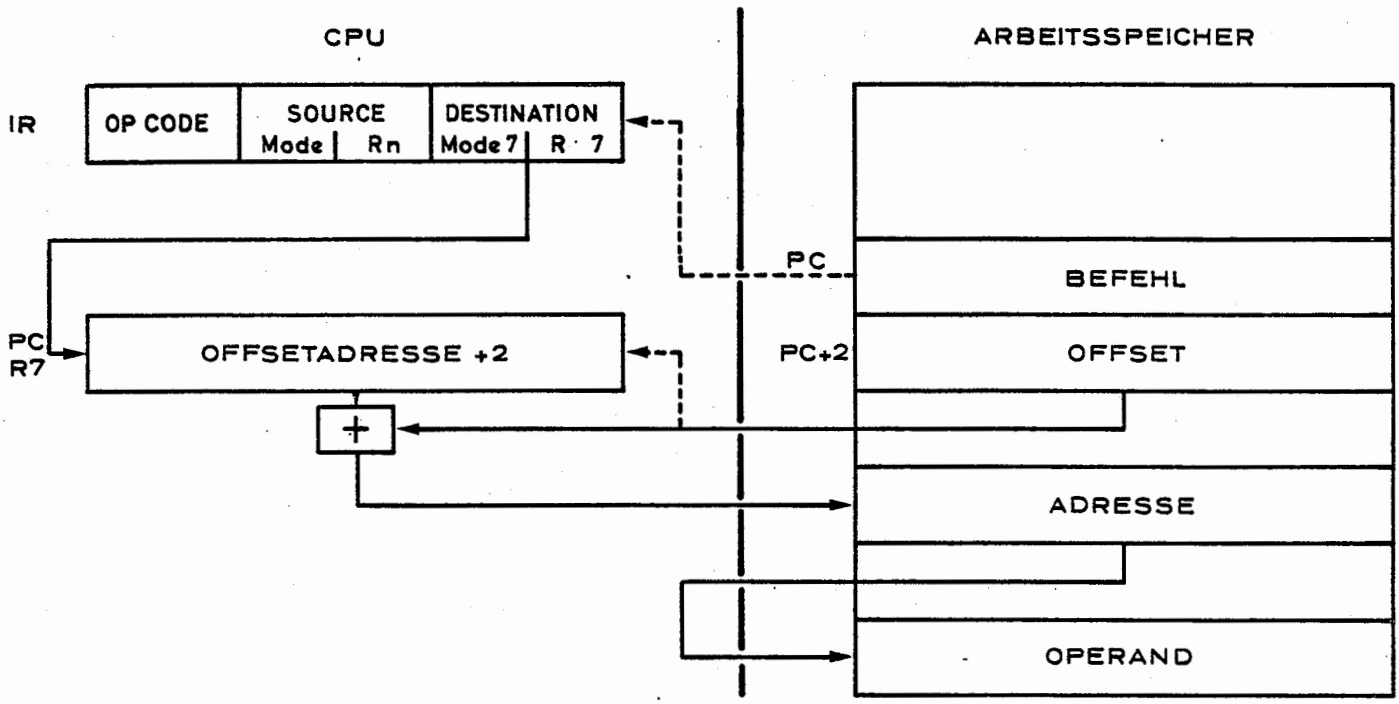
Beispiel zu Mode 6 Reg. 7

symbolisch	oktaler Code	Bedeutung
1000 INC A	005267	Um die Zelle A zu inkrementieren, wird der Inhalt des dem Befehl folgenden Wortes zum Inhalt des PC addiert. daraus ergibt sich die Adresse A= 1060. Der Inhalt dieser Adresse wird dann um 1 erhöht.
1002 54	000054	
1004 -		

3.3.4 Mode 7/Register 7, Relativ Deferred Mode

Hier wird durch Addition von Offsetadresse + Offset + 2 ein Zeiger erzeugt, der auf die Adresse eines Operanden weist.

Symbol: @A



Kurz:

```
    Offsetadresse
+   Offset
+   2
=   Adresse der Adresse eines Operanden
```

Beispiel für Mode 7/Reg. 7

symbolisch	oktaler Code	Bedeutung
CLR @A	005077 000020	Addiere das zweite Wort des Befehls zum PC und erstelle somit die Adresse der Adresse des Operanden. Der Operand wird dann gelöscht.

VOR DER AUSFÜHRUNG			NACH DER AUSFÜHRUNG	
1020	005077	PC	1020	005077
1022	000020		1022	000020
1024			1024	PC
1044	010100		1044	010100
		A=1044		
10100	100001		10100	000000

Arbeitsblatt 3.1 BASIC MODES

Füllen Sie folgende Tabelle unter den gegebenen Voraussetzungen aus. Beachten Sie bitte, daß für jeden Befehl der Liste die selben Voraussetzungen gelten:

(R0)	=	1000	(1000)	=	100	(100)	=	10
(R2)	=	3000	(3000)	=	300	(300)	=	30
(R5)	=	4000	(4000)	=	400	(400)	=	40
(R7)	=	7000	(6504)	=	654	(654)	=	64
(2776)	=	276	(276)	=	26	(26)	=	6
(3500)	=	350	(350)	=	30	(30)	=	0
(4100)	=	410	(410)	=	40	(40)	=	0
(2777)	=	177						

BEFEHL	MODE	ADRESSE DES OPERANDEN	REGISTER INHALT NACH AUSFÜHRUNG	OPERAND		NEUER PC
				VORHER	NACHHER	
INC (R5)	1	4000	4000	400	401	7002
DEC (R2)+						
CLR @ -(R2)						
CLR R5						
INC 100(R5)						
DECB @(R2)+						
CLRB (R5)						
CLRB -(R2)						
INC @1000 (R2)						
CLRB (R0)+						
INC -300(R5)						
INC @ -300(R5)						

Arbeitsblatt 3.2 PC-Modes

Analog zu Arbeitsblatt 3.1

Voraussetzungen:

(R7) = 5000
(1000) = 100
(100) = 10
(10) = 1
(SUM) = 2700, der Offset bei INC SUM soll + 600_8 sein
(LIMIT) = 4000, der Offset bei CLR LIMIT soll - 500_8 sein
(7500) = 1000

BEFEHL	MODE	2.WORT DES BEFEHLS	OPERANDEN ADRESSE	OPERAND	
				VORHER	NACHHER
CLR #1234	27	001234	5002	1234	0
DEC @#1000					
CLR @7500					
CLR 7500					
INC @#100					
CLR 1000					
INC #1234					
INC @1000					
INC @-4004(R7)					
INC SUM					
CLR LIMIT					

KAPITEL 4 - BEFEHLE und PROGRAMMIERTECHNIKEN

4.1 Einführung

Aufgrund der vielfältigen Adressierungsarten verfügt die PDP11 über mehr als 400 Befehle. Hinzu kommt, daß Single- wie Double-Operand-Befehle Daten im Byte- und Wortformat verarbeiten können. Dies spart Speicherplatz und vereinfacht z. B. die Steuerung von Peripherie-Geräten.
Die Bedeutung von Doppel-Operand-Befehlen (typisches Merkmal sogenannter Zwei-Adress-Maschinen) sei an folgendem Beispiel gezeigt:
Der Befehl ADD A,B addiert die Inhalte der Speicherstellen A und B und speichert das Ergebnis in die Zelle B.

Bei Ein-Adress-Maschinen müßte der selbe Vorgang wie folgt programmiert werden:

```
LDA A    lade Inhalt von A in den Akkumulator

ADD B    addiere Inhalt von B auf den Akkumulator

STR B    speichere Inhalt des Akkumulators nach B
```

Der gesamte Befehlssatz der PDP11 läßt sich in folgende vier Gruppen einteilen:

1. Single Operand-Befehle

2. Double Operand-Befehle

3. Verzweigungsbefehle
   - Branch
   - Jump u. Subroutine
   - Trap

4. Sonstige Befehle
   Miscellaneous (HALT, WAIT, NOP, RESET, Memory-Management) und
   Befehle zur Beeinflussung der Condition-Code-Bits.

Die vollständige Auflistung und Beschreibung des Befehlssatzes findet sich im "processor handbook", Kapitel: Instruction Set. Einige wichtige Befehle werden in den folgenden Kapiteln erläutert.

4.2.1 COM(B) und NEG(B)

COM(B) Complement (Single Operand)

Die einzelnen Bits des Operanden werden negiert. Dies entspricht der Bildung des Einer-Komplements bzw. des logischen Kehrwerts.

Beispiel: COM RØ

Vorher	Nachher	
RØ	RØ	NZVC
013333	164444	1001

NEG(B) Negate (Single Operand)

Die einzelnen Bits des Operanden werden negiert und anschließend 1 addiert. Dies entspricht dem Zweier-Komplement bzw. dem arithmetischen Kehrwert, da alle PDP-11en mit Zweierkomplement-Arithmetik arbeiten.

Beispiel: NEG RØ

Vorher	Nachher	
RØ	RØ	NZVC
000010	177770	1001

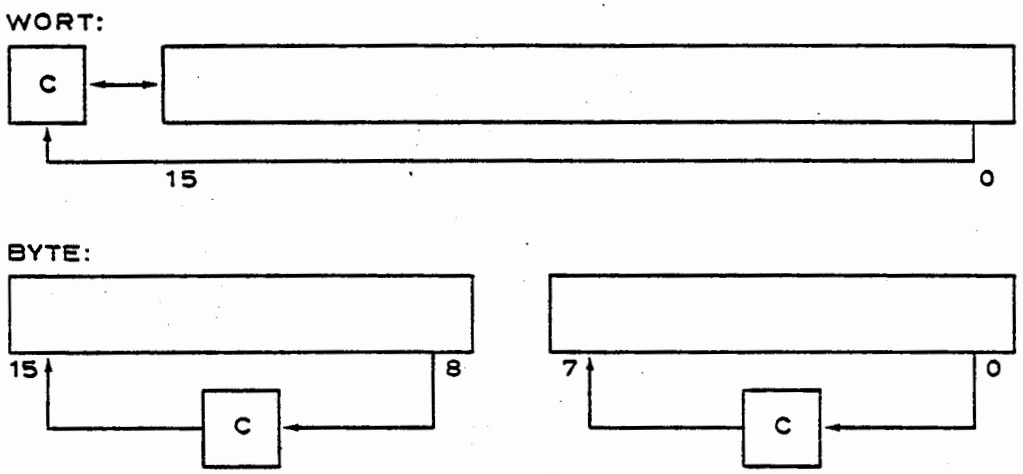
Achtung: Die größte negative Zahl (100000 bei Wortverarbeitung bzw. 200 bei Byteverarbeitung) kann nicht negiert werden, da die entsprechend positive Zahl nicht mehr darstellbar ist. (siehe Zahlenkreis)
In diesem Fall wird das V-Bit (Overflow) gesetzt, um anzuzeigen das beim Negieren ein arithmetischer Fehler aufgetreten ist.

4.2.2 ROR(B), ASR(B) und SWAB

ROR(B) Rotate right (Single Operand)

Verschiebung aller Bits des Operanden um eine Stelle nach rechts. Das C-Bit wird nach Bit 15 (Bit 7) und Bit Ø (Bit 8) ins C-Bit geschoben (≙ Rotation)

Beispiel:

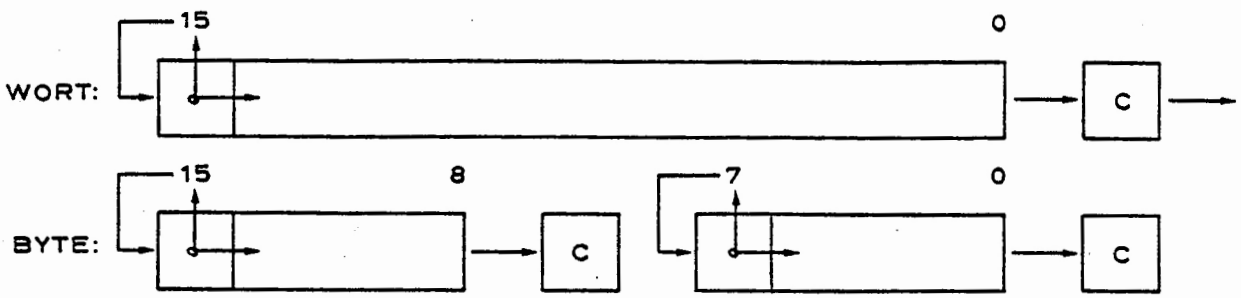


ASR(B) Arithm. Shift right (Single Operand)

Der arithmetische Shift (hier Rechts-Shift) wird auf arithmetische Operanden angewendet, da hier das Vorzeichen berücksichtigt werden muß. Ein ASR(B)-Befehl bedeutet die Division des Operanden durch 2 wobei sich das Vorzeichen selbstverständlich nicht ändern darf (analog: Links-Shift):

Alle Bits werden um eine Stelle nach rechts geschoben. Bit Ø (Bit 8) kommt ins C-Bit, Bit 15 (Bit 7) wird "dupliziert".

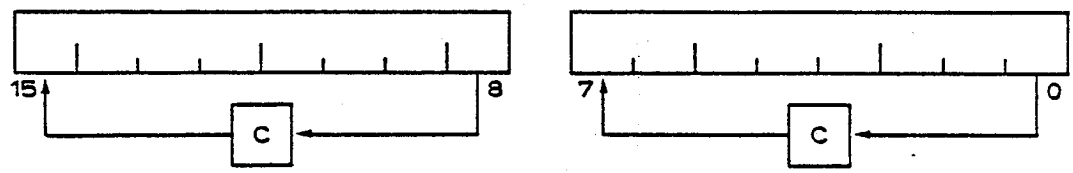
Beispiel:



ROR
RORB

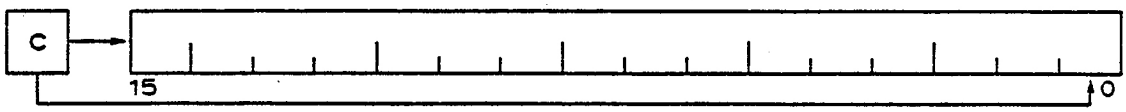
WORT:

BYTE:

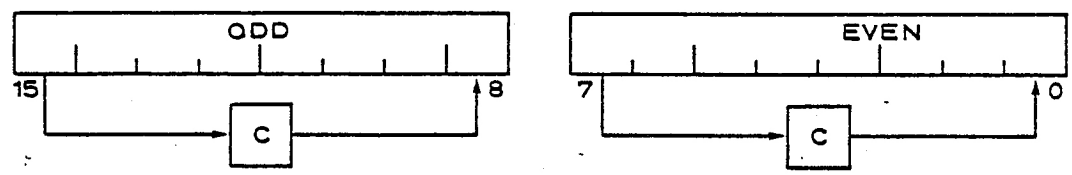


ROL
ROLB

WORT:

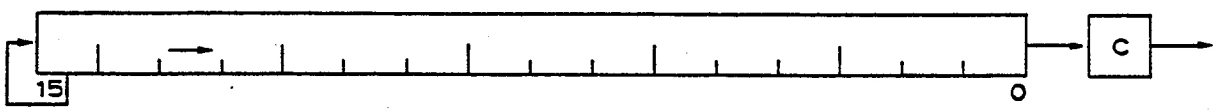


BYTE:

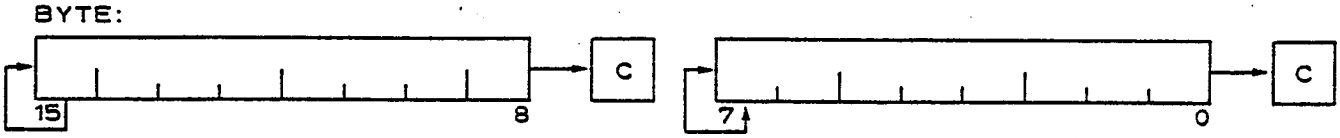


ASR
ASRB

WORT:

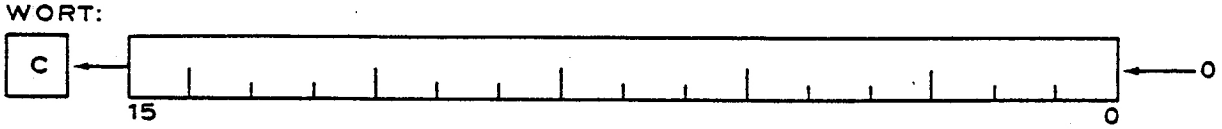


BYTE:

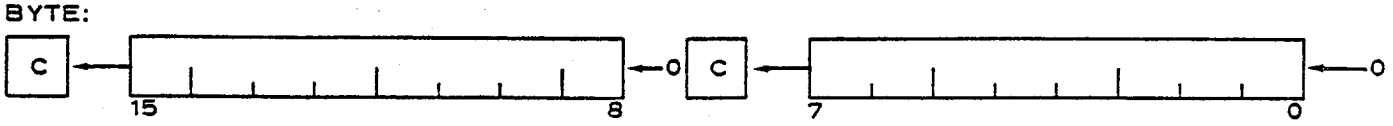


ASL
ASLB

WORT:



BYTE:



SWAB Swap Bytes (Single Operand)

High Byte und Low Byte des Operanden werden vertauscht. Die angegebene Operanden-Adresse muß eine Wortadresse sein. Dieser Befehl erleichtert die Verarbeitung von Datenbytes die andernfalls nur mit 8-fach Shifts möglich wäre.

Beispiel: SWAB R1

Vorher	Nachher	
R1	R1	NZVC
077777	177577	ØØØØ

4.2.3 ADC(B) und SXT

ADC(B) Add Carry (Single Operand)

Der Inhalt des Carry-Bit wird zum Destination-Operanden addiert. Bei Additionen mit mehrfacher Genauigkeit dient dieser Befehl zur Übertragung des Carry (Übertrag) auf den nächsten (höherwertigen) Teiloperanden.

Beispiel:

```
ADD AØ,BØ       Addiere die ersten beiden
                (Teil-)Worte
ADC B1          Addiere C zum nächsthöheren
                (Teil-)Wort
ADD A1,B1       Addiere die beiden höheren
                (Teil-)Worte
```

Damit steht in (B1,BØ) ein 32-Bit Wort

SXT Sign extend (Single Operand)

Abhängig vom Vorzeichen der vorangegangenen Operation (N-Bit) wird der Destination-Operand mit Ø...Ø (N=Ø) bzw. mit 1...1 (N=1) "aufgefüllt".
Damit wird bei Operationen mit mehrfacher Genauigkeit das Vorzeichen auf das nächsthöhere Wort ausgedehnt.

Beispiel: SXT A

Vorher		Nachher	
A		A	
beliebig		177777	(≙ -1)
NZVC		NZVC	
1000		1000	

4.2.4 BIT(B), BIC(B) und BIS(B)

Diese Befehle ermöglichen Operationen auf Bit-Ebene

BIT(B) Bit Test (AND) (Double Operand)

Source- u. Destination-Operand werden logisch "UND"-verknüpft. Das Ergebnis beeinflußt nur die Condition-Code-Bits des Prozessorstatusworts. Die Operanden bleiben unverändert!

Beispiel: BIT #30,R3 Test Bits 3 u. 4 von R3

Ergebnis: -Z = 1, N = 0 wenn Bit 3 = Bit 4 = Ø (von R3) d.h. die Verknüpfung ergibt Ø

-Z = Ø, N = Ø wenn Bit 3 = Bit 4 = 1, od.Bit 3 ≠ Bit 4

-N = 1 wenn das höchste Bit des Ergebnisses gesetzt wird

Stets gilt: V = Ø und
C wird nicht beeinflußt

BIC(B) Bit Clear (Double Operand)

Es wird jedes Bit des Destination-Operanden gelöscht, welches mit einem gesetzten Bit des Source Operanden übereinstimmt.

Beispiel: BIC R3,R4

	Vorher	Nachher
Source	R3 001234	R3 001234
Destin.	R4 000111	R4 000101
		NZVC 000-

BIS(B) Bit Set (OR) (Double Operand)

Source- u. Destination-Operand werden logisch "OR"-verknüpft. Jede Bitposition der Source welche "1" ist erwirkt das Setzen der selben Bitposition der Destination.

Beispiel: BIS RØ,R1

	Vorher	Nachher
	RØ	RØ
Source	001234	001234
	R1	R1
Destin.	000111	001335
		NZVC
		0000

4.2.5 CMP(B) UND TST(B)

CMP(B) Compare-Befehl (Double Operand)

Der CMP-Befehl stellt ein wichtiges Hilfsmittel für Branchbefehle dar. Er vergleicht den Source- mit dem Destination-Operanden. Das Ergebnis dieses Vergleichs schlägt sich ausschließlich im Prozessor-Statuswort (PSW) nieder, d.h. die Operanden selbst werden nicht verändert.

Beispiel: Sei RØ = 2
CMP #5,RØ : 5 - 2 = + 3
Source Dest.

PSW 0 0 0 0
N Z V C

TST(B) Test-Befehl (Single Operand)

Falls aufgrund eines bereits vorhandenen (früher berechneten) Wertes verzweigt werden soll, kann auf diesen der TST-Befehl angewendet werden.
Er veranlaßt das Setzen der Bits N und Z im PSW abhängig davon ob der Wert null, negativ oder positiv ist. Bei einem positiven Wert ist Z = N = Ø ("nicht null und nicht negativ = positiv")

Beispiele zu CMP und TST siehe Kapitel über Branch-Befehle.

Anmerkung zu BYTE-Instruktionen

- Bei Byte-Instruktionen ist immer Bit 15 des Instruktionscodes gesetzt.

- Es ist dadurch die Adressierung von einzelnen Bytes (Halbwörtern) möglich.

- Byte-Instruktionen in Verbindung mit AUTOINCREMENT-und AUTODECREMENT-Adressierung verändern den Inhalt des bezogenen Registers nur um 1.

- Byte-Instruktionen in Verbindung mit AUTOINCREMENT DEFERRED und AUTODECREMENT DEFERRED Adressierung verändern den Inhalt des bezogenen Registers jeweils um 2 (der Inhalt des Registers ist die Adresse einer Adresse, die immer ein Wort mit 16 Bits belegt).

- Byte-Instruktionen in Verbindung mit den Registern 6 und 7 verändern diese Register immer um 2.

- Byte-Instruktionen in Verbindung mit dem REGISTER MODE adressieren das LOW BYTE eines Registers.

- Die Instruktion MOVB in ein Register (Destination im Register-Mode) füllt das obere Byte des Registers mit dem Vorzeichen des Source-Operanden auf (Sign-Extension).

Arbeitsblatt 4.1

1. Wie sieht der Maschinencode für folgende Befehle aus?

   a) Addiere 3 auf den Inhalt der Adresse 1100:

   b) Lösche den Inhalt der Adresse 1400:

   c) Bringe den Inhalt der Adresse 100 in die Adresse 5000:

2. Wie wird das PSW nach folgenden Befehlen gesetzt?

	SS	DD	N	Z	V	C
a) Addieren	(-2),	(-1)				
b) Subtrahieren	(-2),	(-1)				
c) Vergleichen	(-2),	(-1)				

Arbeitsblatt 4.2

1. Mit welchem Befehl kann man das Bit 7 in R4 löschen?

2. Mit welchen Befehlen kann man den Inhalt von R1 mit 4 multiplizieren?

3. Wie werden die Condition-Bits nach Ausführung dieser Befehle gesetzt?

	N	Z	V	C
MOV #177543,RØ				
BIT #200,RØ				

4. Die folgenden Befehle sollen in der angegebenen Reihenfolge ausgeführt werden.

   Was ist der Inhalt von Register Ø nach Ausführung dieser Befehle?

	Inhalt RØ
a) MOV #42345,RØ	_____
b) SWAB RØ	_____

5. Wie kann man feststellen, ob Bit 5 in RØ gesetzt ist?

4.2.6 BRANCH-Befehle

Diese Befehle ermöglichen Programm-Verzweigungen. Man unterscheidet zwei Arten:

a) Der Verzweigungsbefehl ist unbedingt (BRANCH)

b) Der Verzweigungsbefehl ist bedingt. Die Entscheidung ob verzweigt wird hängt vom Zustand der Condition-Code-Bits im Prozessorstatuswort (PSW) ab.

Die Zieladresse eines Branch ergibt sich aus der Summe von OFFSET und Program-Counter. Unter dem OFFSET versteht man die Anzahl der Worte vom augenblicklichen PC an gerechnet bis zur Zieladresse. Es ist zu beachten, daß der PC bereits zum nächsten Wort zeigt, also zum Befehl, der dem Branch folgt (=updated PC).

Der PC kann keine Byte-Adresse ausdrücken, der OFFSET ist somit immer in Worten angegeben. Der OFFSET wird automatisch mit zwei multipliziert und dann zum PC addiert. Bit 7 des Offset ist das Vorzeichenbit. Ist dieses Bit gesetzt, so ist der Offset negativ und die Verzweigung erfolgt in Rückwärts-Richtung.

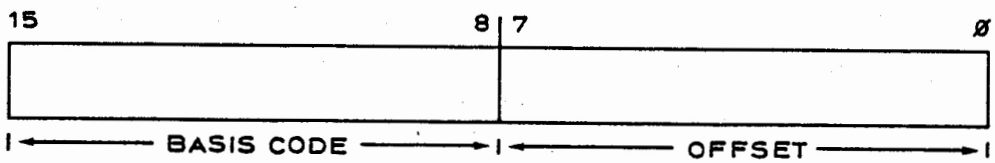
Mit dem 8-bit Offset ist eine Verzweigung in Rückwärts-Richtung um 200_8 Worte (400 Bytes) vom augenblicklichen PC an möglich. In Vorwärts-Richtung beträgt der Wert 177_8 Worte (376 Bytes) - ebenfalls vom augenblicklichen PC an gerechnet.

OFFSET - Formel

$$\text{OFFSET} = \frac{\text{ZIELADRESSE - UPDATED PC}}{2}$$

Der OFFSET wird im Befehlscode des Branch angegeben

Befehlsformat



Es ist zu beachten, daß in der Programmier-Karte bzw. im Prozessor-Handbuch immer nur der Basis Code des Branch angegeben ist, wobei die OFFSET-Bits alle Ø sind.

Einteilung

Die Sprungbefehle lassen sich ihrer Verwendung nach in 3 Gruppen einteilen:

1. Unbedingte und einfache bedingte Verzweigungen

BR, BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BCS

Diese Befehle testen jeweils ein Bit des PSW daraufhin ob es gesetzt oder gelöscht ist (Ausnahme BR).

Beispiele

BR Branch (unbedingte Verzweigung)

Operation: PC = PC + 2 * Offset

Beschreibung: ohne Bedingung wird das Programm innerhalb des Bereichs von +127 und -128 Worten fortgesetzt.

Beispiel:

```
1000       BR  A      ; meist wird der Offset
1002        ...       ; symbolisch angegeben
1004        ...       ; hier Branch nach A
1006 A:     ...
1010        ...
 .
 .
 .
```

Berechnung des Offset nach der Formel: $\frac{1006 - 1002}{2} = 2$

Der oktale Offset beträgt also 2 und BR A muß als 000402 programmiert werden.

Anmerkungen: Für kurze Sprünge ist die Offset-Formel zu unhandlich und man behilft sich durch Abzählen der Speicherstellen, wobei beim Branchbefehl mit -1 (!) zu beginnen ist.

BPL Branch if plus (verzweige, falls positiv)

Operation: PC = PC + (2 x Offset), falls N = 0

Beschreibung: Testet das N-Bit (Negativ) und bewirkt einen Branch wenn N=0 (≙nicht negativ≙positiv)

BMI Branch if minus (verzweige, falls negativ)

komplementär zu BPL: Branch, falls N=1

BNE Branch if not equal (to Ø) (verzweige, falls nicht null)

Operation: PC = PC + (2 x Offset), falls Z=0

Beschreibung: Testet das Z-Bit (Zero) und bewirkt einen Branch wenn Z=Ø (≙ nicht null)

BEQ Branch if equal (to Ø)

komplementär zu BNE: Branch, falls Z=1

Programmbeispiel:

Zähle von 1000_8 abwärts bis Ø

```
1000       MOV #1000,R4          ; lade 1000 nach R4
1002       -                     ;
1004  A:    DEC R4                ; erniedrige R4 um 1
1006       BNE A                 ; springe nach A
1010       HALT                  ; solange Z=Ø
```

2. Verzweigungen für 2er Kompl.-Arithmetik

BGE, BZT, BGT, BLE

Hier werden Kombinationen von Z,V,N getestet. Damit sind Vergleiche von arithmetischen Werten möglich.

Beispiele

BGE Branch if greater than or equal (to Ø)

Operation: PC = PC + (2 x Offset), falls N¥V=0

Beschreibung: Es wird verzweigt, falls die Bits N und V beide gelöscht oder beide gesetzt sind. Dies ermöglicht eine korrekte Verzweigung auch bei arithmetischem Überlauf.
(Man mache sich dies mit Hilfe des CMP-Befehls und des Zahlenkreises klar)

BLT Branch if less than (Ø)

komplementär zu BGE, Branch falls N¥V=1

Programmbeispiele:

Zähle von 0 bis 1000_8

```
1000       CLR R4              ; lösche R4
1002  A:   INC R4              ; erhöhe R4 um 1
1004       CMP R4,#1000        ; vergleiche R4 mit 1000
1006       -                   ; d.h. (R4) -1000 = ?
                               ;  N-Bit
1010       BLT A               ; solange R4<1000 wird
                               ;  verzweigt
1012       HALT                ; d.h. solange N=1 ist
```

3. Verzweigungen ohne Berücksichtigung des Vorzeichens

BHI, BLOS, BHIS, BLO

Hier kann aufgrund absoluter -also vorzeichenunabhängiger- Operationen verzweigt werden, z.B. bei Vergleich von Adressen.

Beispiele

BHI Branch if higher

Operation: PC = PC + (2 x Offset), falls C=Ø und Z=Ø

Beschreibung: C=Ø und Z=Ø tritt bei einem CMP-Befehl auf, wenn der Source-Operand betragsmäßig (ohne Vorzeichen) größer ist als der Dest.-Operand.

BLOS Branch if lower or same

komplementär zu BHI, Branch falls C∨Z=1

Programmbeispiel: Bringe in RØ ein Bit auf vorgegebene Position (R4)

```
1000        MOV #2000,R4        ; lade 2000 nach R4
1002        2000                ; (Bit 10=1)
1004        CLR RØ              ; lösche RØ und erhöhe
1006        INC RØ              ; RØ um 1 (Bit Ø=1)
1010   A:   ROL RØ              ; verschiebe RØ um 1
                                ; Position nach links
1012        CMP R4,RØ           ; vergleiche RØ mit R4
                                ; d.h. RØ-R4=? Z=?
1014        BHI A               ; springe, solange R2
                                ; größer ist.
1016        HALT
```

Arbeitsblatt 4.3

Setzen Sie den richtigen Branch-Befehl ein. Die Inhalte der Register sind keine Adressen.

Befehl	Es soll verzweigt werden wenn
CMP R0,R1	(R0)=(R1)
_____	
CMP R0,R1	(R0)>(R1)
_____	
CMP R0,R1	(R0)≤(R1)
_____	
CMP R0,R1	(R0)≥(R1)
_____	
CMP R0,R1	(R0)≠(R1)
_____	

4.2.7 JUMP

Mit dem JUMP-Befehl kann man, im Gegensatz zu dem Branchbefehlen, keine Bedingungen abfragen. Der Vorteil beim JUMP liegt darin, daß man eine volle 16-Bit-Adresse als Sprungziel angeben kann, in der Sprungweite also nicht beschränkt ist.

Die Angabe einer solchen absoluten Zieladresse birgt jedoch den Nachteil in sich, daß eine Verschiebung des Programms nicht ohne Änderung dieser Adressen möglich ist. Darin aber liegt gerade der Vorteil der Branchbefehle, dessen Zieladressen sich stets relativ zum PC des Programms errechnen. Somit bleibt die Lage der Zieladresse relativ zum Programm immer konstant, unabhängig von der Lage des Programms selbst (=PIC=position independent coding).

4.3 Ein-/Ausgabe - Programmierung (ohne Interrupt)

4.3.1 Einführung

Jeder Verkehr zwischen einem Programm und der Außenwelt (Peripherie) geschieht über den UNIBUS. In den meisten Fällen befinden sich an der Schnittstelle (Interface) zwischen UNIBUS und Peripheriegerät ein oder mehrere Register, so daß die prinzipiellen Betrachtungen für jeden programmierten Datenverkehr am Beispiel eines Terminals (TTY, LA 36,LA120, VT100 o.ä.) angestellt werden können.
Das Interface-Modul DL11 erlaubt die serielle und asynchrone Übertragung von ASCII-Zeichen zwischen Druckerteil bzw. Tastatur einerseits und UNIBUS (CPU,Speicher) andererseits.
Ein serielles Interface hat fünf grundlegende Funktionen zu erfüllen:

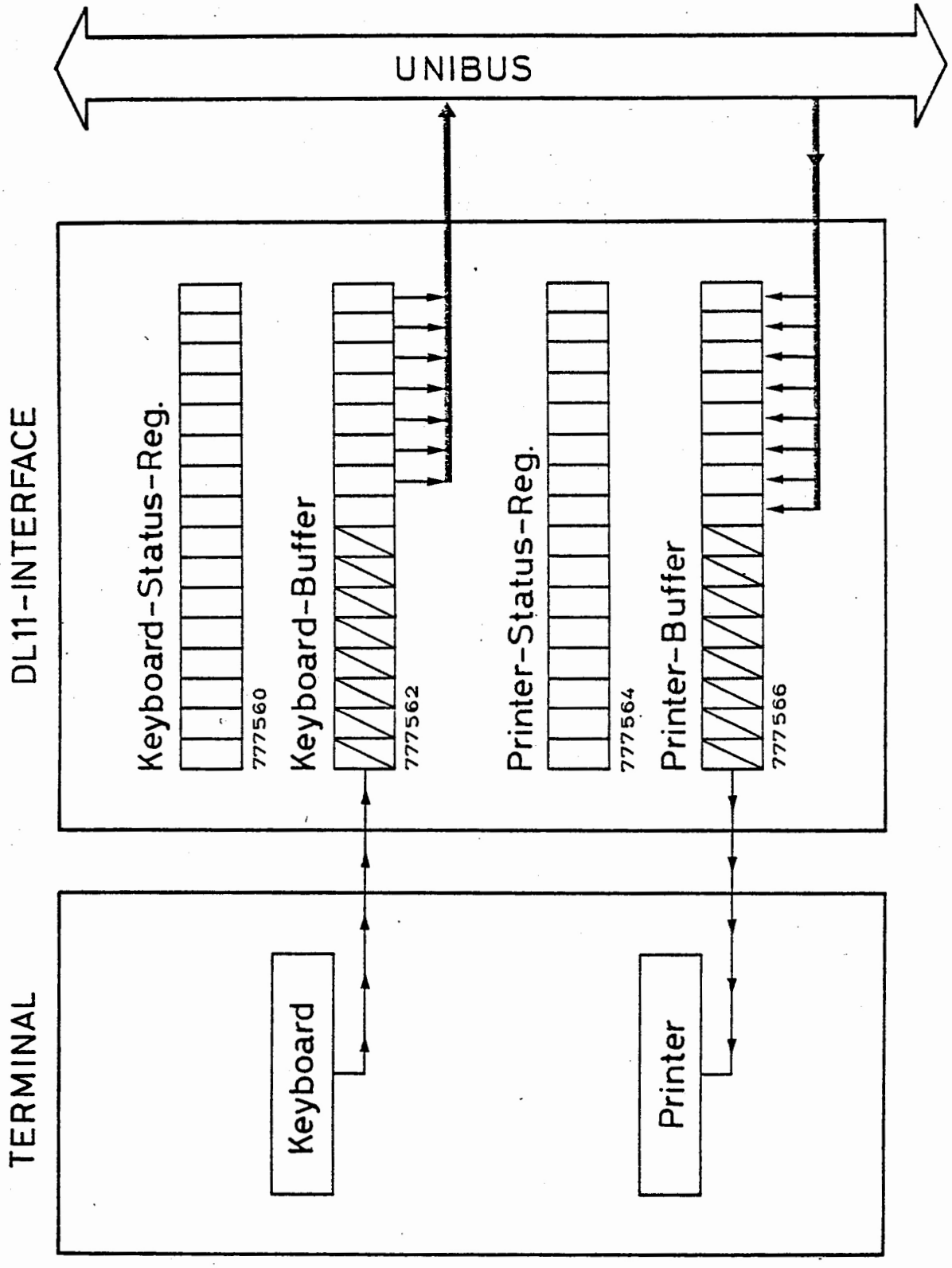
1. Erkennen der angewählten Geräte-Adresse
2. Erzeugen der Interrupt-Signale an den Prozessor
3. Kontrolle und Überwachung der Peripherie-Operation
4. Datenzwischenspeicherung (Data-Buffer)
5. Umsetzung: parallele <-> serielle Daten

Das DL11 besitzt für jede Datenübertragungsrichtung zwei Register:

1. Ein Keyboard-Status-Register und ein Keyboard-Buffer für die Transferrichtung: Tastatur (Keyboard) -> UNIBUS
2. Ein Printer-Status-Register und ein Printer-Buffer für die Transferrichtung: UNIBUS -> Drucker (Sichtgerät)

Folgendes Bild vermittelt einen Überblick

TERMINAL-INPUT/OUTPUT

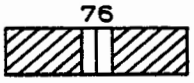
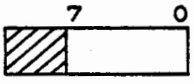
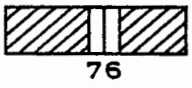
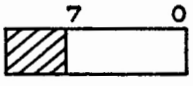


4.3.2 Geräteregister

Wie bereits in früheren Kapiteln beschrieben, sind die Register von Geräten wie Speicherzellen ansprechbar. Die Adressen liegen in den obersten 4K des Adressbereichs, also zwischen 760000 und 777776 (=I/O-Page).

Es ist zu beachten, daß nicht alle Geräteregister (-Bits) les- und schreibbar sind. So ist es z.B. nicht möglich (und nicht sinnvoll) das Keyboard-Buffer vom UNIBUS aus zu beschreiben.

Die Register des DL11 (Terminal-Interface):

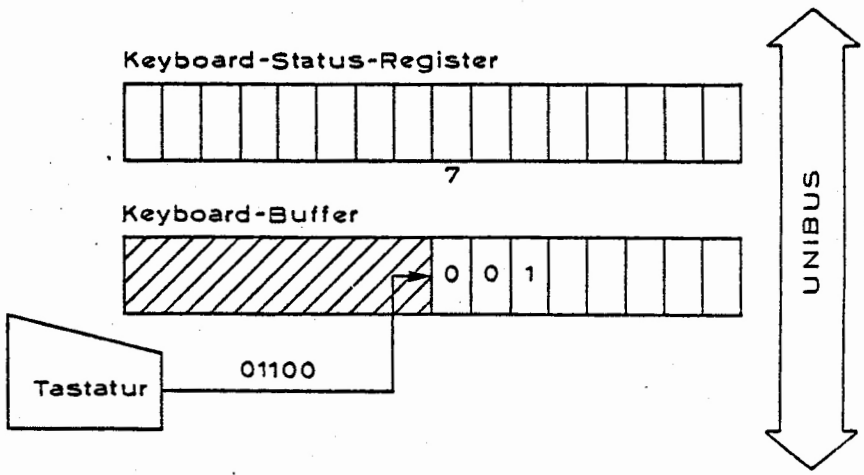
Register	Adresse (physikal.)	Format	Funktionen
Keyboard-Status (KBS)	777560		7 -Done Bit- gesetzt, wenn ein neues Zeichen im KBB vorliegt;gelöscht, wenn das KBB gelesen wird. 6 -Interrupt Enable Bit- kann vom Programm gesetzt werden. Dadurch wird ein Interrupt ermöglicht.
Interrupt-Vektor	60		
Keyboard-Buffer (KBB)	777562		0-7 Enthält ein Zeichen vom Keyboard im ASCII-Code und das Parity-Bit.
Printer-Status (PTS)	777564		7 -Ready Bit- es wird gesetzt nach Beendigung einer Druckoperation;gelöscht, wenn ein neues Zeichen im PTB vorliegt. 6 -Interrupt Enable Bit- kann vom Programm gesetzt werden. Dadurch wird ein Interrupt ermöglicht.
Interrupt-Vektor	64		
Printer Buffer (PTB)	777566		0-7 Enthält ein Zeichen, das auf den Drucker gebracht wird.

4.3.3 DATENEINGABE

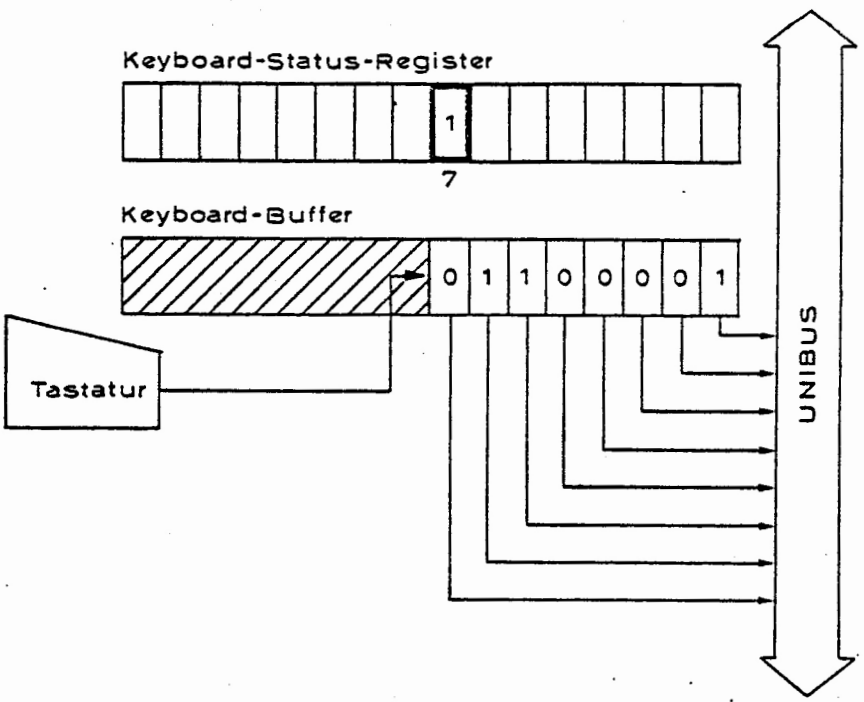
Jedes eingegebene Zeichen wird einzeln behandelt. Ein Zeichen wird in den Buffer gebracht und von dort per Programm geholt.

Vorgang

1	Es wird auf eine Taste gedrückt.
2	Das Zeichen wird in ASCII-Code übersetzt.
3	Der Code wird seriell übertragen.



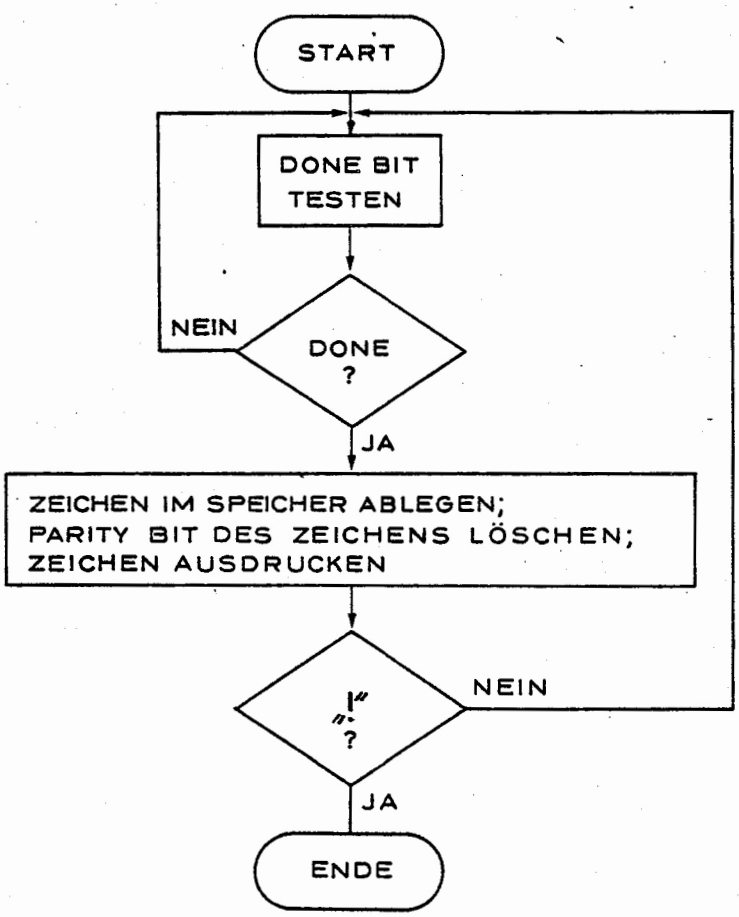
4	Sobald das Register voll ist (das ganze Zeichen wurde übertragen) wird das Done Bit (7) gesetzt.
5	Das Done Bit bleibt solange gesetzt, bis das Zeichen per Befehl geholt wird
6	Die Übertragung vom Buffer auf den UNIBUS geschieht Bitparallel.



Dateneingabe-Programmierung (ohne Interrupt)

Wegen der unterschiedlichen Geschwindigkeiten der seriellen Übertragung von Terminals und der parallelen Übertragung auf den UNIBUS muß erst per Programm abgefragt werden, ob ein Zeichen im Keyboard Buffer vorhanden ist. Dies geschieht durch Testen des "Done Bits".

Im folgenden Programm sollen Zeichen vom Keyboardbuffer (KBB) geholt und in den Speicher abgelegt werden. Das jeweilige Zeichen soll auch auf dem Printer erscheinen (Echo). Mit Drücken des Zeichens "!" soll das Programm beendet werden.

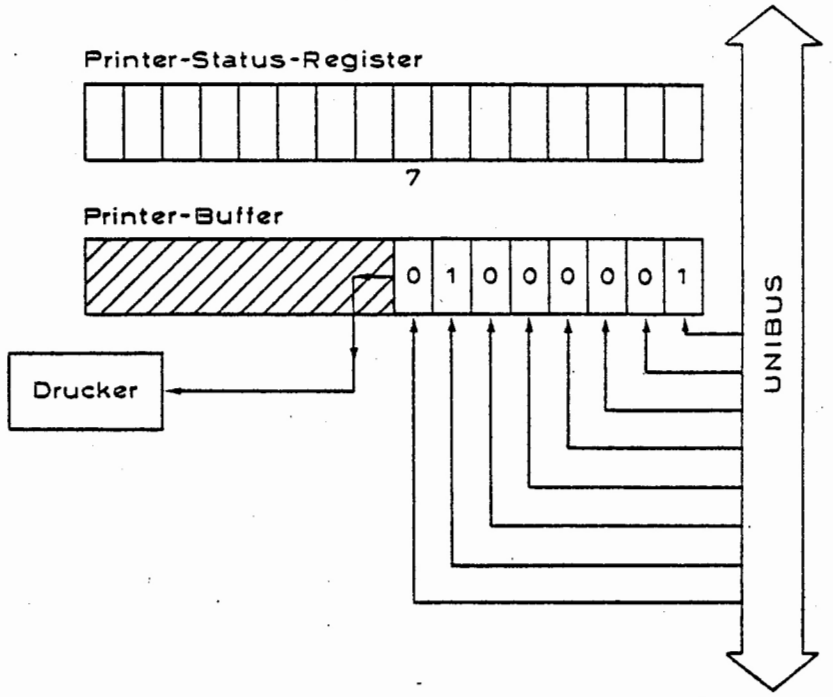
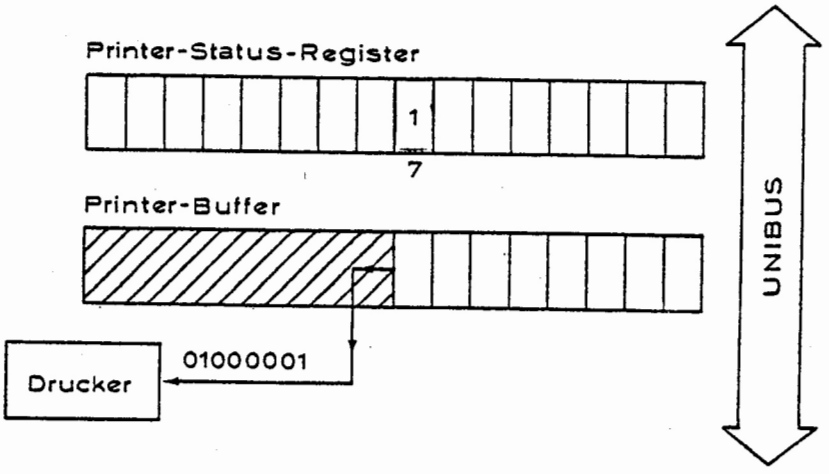


Programm siehe Anhang A.2

4.3.4 DATENAUSGABE

Jedes Zeichen wird einzeln geholt und in das Buffer-Register geladen. Von dort wird es auf den Drucker gebracht.

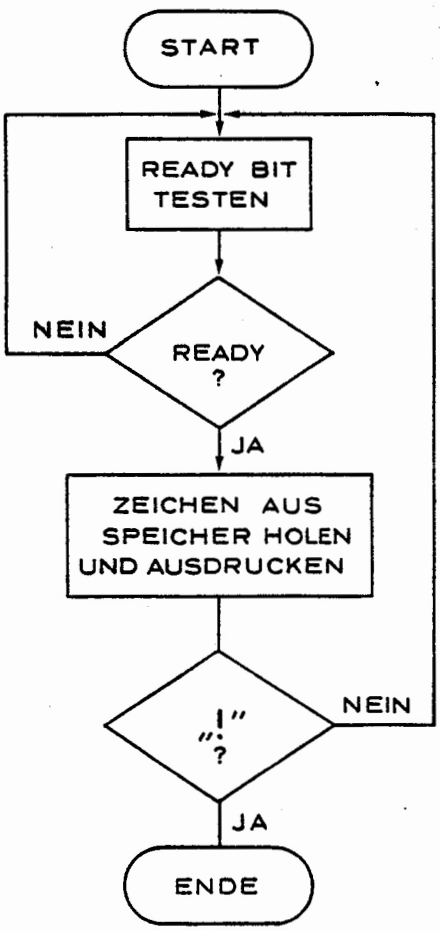
Vorgang

1	Ein Zeichen wird per Programm in den Printer-Buffer gebracht.	
2	Das Zeichen wird seriell aus dem Buffer auf den Drucker übertragen.	
3	Sobald das Register entleert ist, wird das Ready Bit gesetzt.	
4	Das nächste Zeichen kann ausgegeben werden.	

Datenausgabe-Programmierung (ohne Interrupt)

Ein Zeichen wird viel schneller ins Printer Buffer gebracht, als es mechanisch vom Drucker abgearbeitet werden kann. Deshalb muß erst per Programm festgestellt werden, ob das Zeichen vollständig abgearbeitet wurde. Die Prüfung erfolgt durch Testen des "Ready Bits".

Das folgende Programm soll im Speicher abgelegte ASCII-Zeichen ausdrucken. Erscheint das Zeichen "!" so wird das Programm angehalten.



Programm siehe Anhang A.2

Arbeitsblatt 4.4

1. Was sind die fünf grundlegenden Funktionen eines Interfaces?

   .....

   .....

   .....

   .....

   .....

2. Mit welchem Befehl kann man prüfen, ob das Done Bit im Keyboard Status Register gesetzt ist?

3. Welche Wirkung haben diese Befehle?

```
A:  BITB # 200,@#1777564
    BPL  A
```

4. Was ist ein Echo?

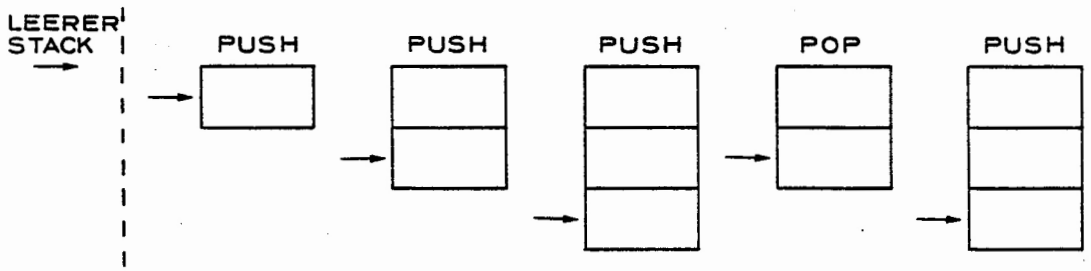
5. Schreiben Sie eine Routine, die Ihren Namen ausdruckt.

4.4 STACK, TRAP, INTERRUPT

4.4.1 STACK-Technik

Der Stack ist ein spezieller Speicherbereich, der entweder vom Programmierer oder von der Systemhardware zum vorübergehenden Ablegen von Daten und Adressen verwendet wird.

Der Stack arbeitet nach dem "last in, first out"-Konzept (LIFO). Ein Beispiel für dieses Prinzip ist ein Sackbahnhof: Die Wagen, die zuletzt einfuhren, fahren zuerst wieder hinaus. Stack: Das zuletzt eingespeicherte Wort wird als erstes wieder abgeholt.



Push: 1 Wort auf Stack ablegen
Pop: 1 Wort vom Stack holen

Eine solche Stackorganisation kann programmiert werden, indem man sich des auto-increment/-decrement-Modes mit einem beliebigen Register RØ bis R6 bedient.

Das Register R6 hat dabei eine besondere Bedeutung: Einerseits kann es vom Programm benutzt werden, andererseits wird es von der Hardware bei Trap und Unterprogrammen automatisch verwendet. R6 wird deshalb als Stackpointer (SP) bezeichnet. Wie aus obigem Bild ersichtlich, zeigt der Stackpointer (Pfeil) immer auf das zuletzt eingeschriebene Wort.

Der Stack wird bei der PDP-11 von hohen Adressen nach unten beschrieben. Dieser Vorgang wird "Push" genannt. Man bedient sich dazu des auto-decrement Modes:

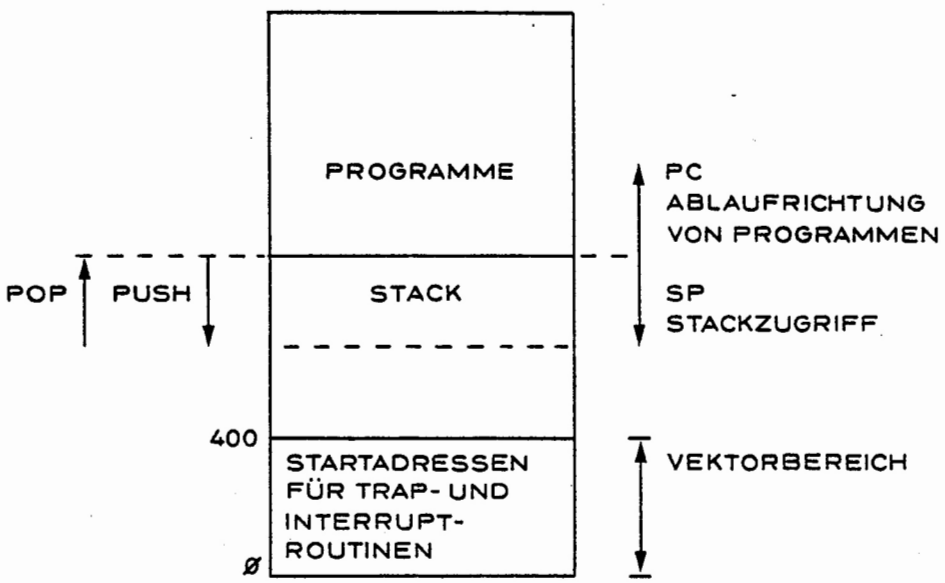
```
"Push":   MOV Source, -(SP)      ; lade den Source Operanden
                                   dorthin, wohin SP-2
                                   zeigt.
```

Das Auslesen aus dem Stack nennt man "Pop". Man bedient sich in analoger Weise des auto-increment Modes.

```
"Pop":    MOV (SP)+, Destination ; lade den Wert auf den
                                   SP zeigt nach "Desti-
                                   nation" und erhöhe SP
                                   anschließend um 2.
```

Nach einem Pop ist der SP um 2 erhöht und zeigt auf die nun zuletzt beschriebene Speicherzelle.

Üblicherweise wird ein Programm in den höchsten Speicherbereich geladen. Oft ist es von Vorteil, den Stack gleich darunter anzuordnen. Damit vermeidet man ein Überschreiben des Programmes bei Stack-Routinen.



Der Stack darf nur bis zum Speicherplatz 400 benutzt werden, da der Vektorbereich wichtige Informationen beinhaltet, welche nicht überschrieben werden dürfen! Jeder Versuch, mit dem Stack Pointer unter die Zelle 400 (resp. Stack Limit) zu schreiben, hat einen Trap nach Zelle 4 zur Folge (Stack Overflow Trap).

Achtung: Der Stack Overflow Trap verwendet seinerseits den Stack!
(Genaueres siehe Processor Handbook)

Die Größe des Stack ist stark programmabhängig. Er sollte deshalb in jedem Programm genau vorausberechnet werden, damit die Adresse 400 nicht unterschritten wird.

4.4.2 TRAP

"TRAP" ist der Oberbegriff für Programmunterbrechungen hervorgerufen durch:

Interrupt	(Anforderung von externen Geräten)
Programm	(programmierte Unterbrechnung)
Hardware	(CPU-Hardware erkennt Fehler)

Jeder TRAP löst automatisch folgenden Vorgang aus:

1. PS_{alt} ⟶ -(SP)
2. PC_{alt} ⟶ -(SP)
3. (Vektor) ⟶ PC
4. (Vektor +2) ⟶ PS

1. Der Prozessorstatus des unterbrochenen Programms wird auf den Stack gelegt.
2. Der Programm-Counter des unterbrochenen Programms wird auf den Stack gelegt (=Rückkehradresse).
3. Eine durch die Art des TRAP definierte Adresse (Vektor) wird ausgegeben. Der Inhalt dieser Adresse wird in den Programm-Counter (PC) geladen (=Startadresse für Bedienungsprogramm).
4. Der Prozessor erhöht den Vektor um 2 und verwendet diese Adresse um den neuen Prozessorstatus zu laden (=PS für Bedienungsprogramm).

Beispiel für eine TRAP-Sequenz

```
       4   001012
       6   000340

    1000   012706   START:   MOV #1000,SP
           001000
    1004   005037            CLR @#160000
           160000
    1010   000000            HALT

    1012   000240   ERROR:   NOP
    1014   000000            HALT
    1016   000002            RTI
```

TRAP

PSW	⟶	-(SP)	XXXXXX	⟶	<776>
PC	⟶	-(SP)	1010	⟶	<774>
(VECT)	⟶	PC	1012	⟶	PC
(VECT+2)	⟶	PSW	340	⟶	PSW

STACK

```
000776   XXXXXX
000774   001010
000772
000770
```

RTI

(SP)+	⟶	PC	1010	⟶	PC
(SP)+	⟶	PSW	XXXXXX	⟶	PSW

Der Fehler in Zelle 1004 (welcher?) löst einen TRAP durch 4 aus. Die Bedeutung des Befehls RTI (Return from Interrupt) wird in Kap. 4.4.3 erklärt.

TRAP-CATCHER

```
        4   000006
        6   000000              HALT

     1000   012706    START:    MOV #1000,SP
            001000
     1004   005037              CLR @#160000
            160000
     1010   000000              HALT
```

TRAP

PSW	⟶	-(SP)	XXXXXX	⟶	<776>
PC	⟶	-(SP)	1010	⟶	<774>
(VECT)	⟶	PC	6	⟶	PC
(VECT+2)	⟶	PSW	0	⟶	PSW

STACK

```
000776   XXXXXX
000774   001010
000772
000770
```

Hinweis:

Jede vernünftige Programmierung muß mit dem Aufsetzen (=Laden) des Stackpointers und der Trap-Vektoren beginnen, da man zu keinem Zeitpunkt vor Fehlern des Programms bzw. der Hardware sicher ist. Nicht definierte Stackpointer u. Vektoren führen bei Traps u.U. zur Zerstörung des Programms und machen eine effiziente Fehlersuche praktisch unmöglich! Sieht man keine Serviceroutinen vor, so muß zumindest ein Trap-Catcher aufgesetzt werden!

4.4.3 INTERRUPT

1. Definition und Anwendung

- Unter Interrupt vesteht man die Unterbrechnung eines laufenden Programmes durch ein bestimmtes externes Ereignis.

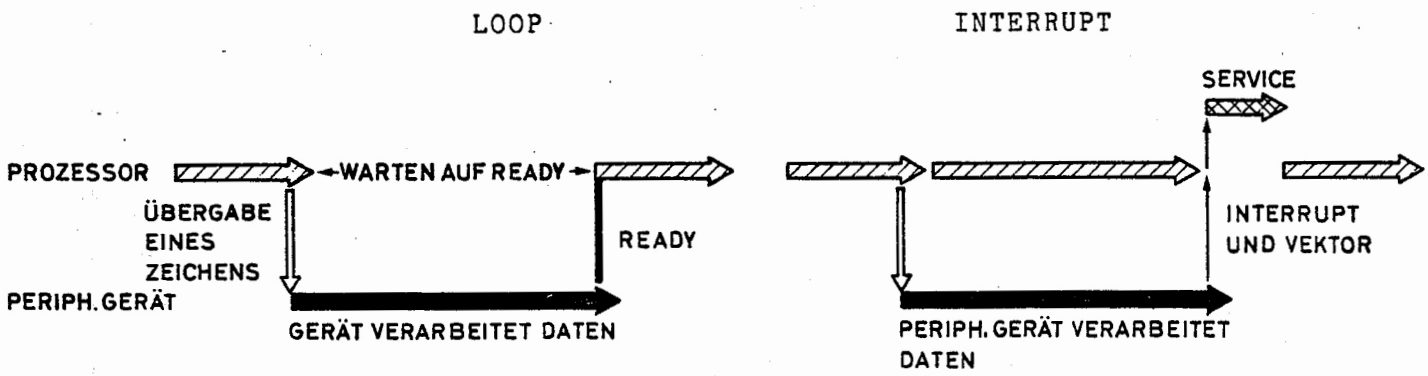
- Interrupt-Steuerung wird dort angewendet, wo das Eintreffen eines Ereignisses das laufende Programm unterbrechen soll.

- Die Interrupt-Steuerung erlaubt es, die Kapazität eines Rechners optimal und zielgerichtet auszunützen.

Beispiel:

Die Ausgabe eines Zeichens zum Terminal benötigt ca. 100 ms. Die Operationen der CPU belegen davon ca. 20 us, d.h. ca. 0,2‰. Während der restlichen 99,98 % der Ausgabezeit wartet die CPU in einer Schleife bis das Ready-Bit gesetzt ist. Wird die Ausgabe interruptgesteuert, kann der Prozessor während dieser Zeit von 99,9 ms ca. 20'000 Befehle weiterer Programme abarbeiten. Sobald das Zeichen ausgeben wird, setzt die Hardware des Terminal-Interface das Ready-Bit, was sogleich einen Interrupt Request für die Ausgabe des nächsten Zeichens erzeugt. Obwohl der Prozessor für weitere Aufgaben eingesetzt wurde, verzögert sich der Ausgabevorgang von außen gesehen nicht.

Vergleich

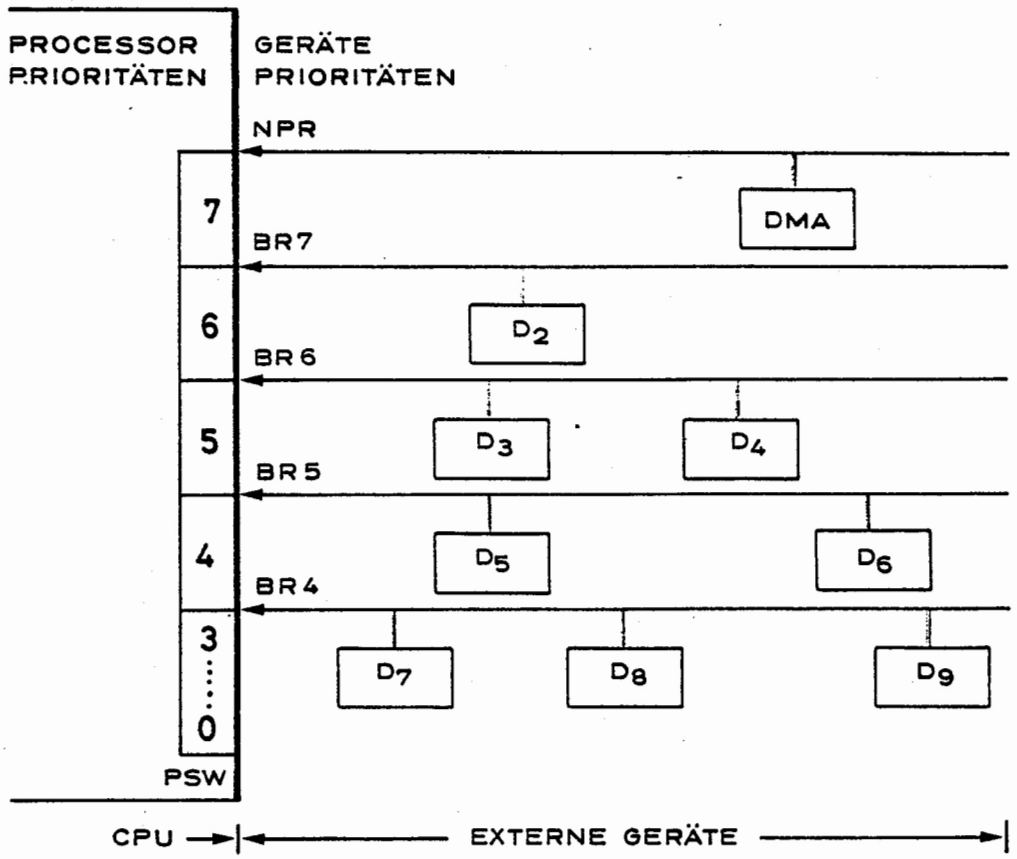


Da meist mehrere Geräte "interrupt-fähig" sind übergibt das unterbrechende Gerät einen Vektor um den Prozessor auf die entsprechende Interrupt-Serviceroutine zu verweisen.

2. Das Prioritätenschema der PDP 11

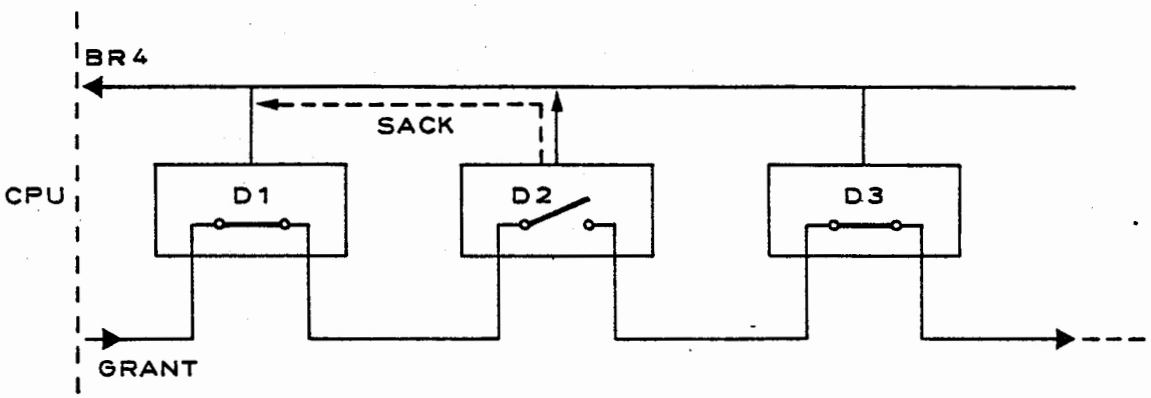
A. Vertikale Priorität

Da alle Pheripherals und der Prozessor den Unibus als gemeinsamen Datenkanal benutzen und somit die Gefahr besteht, daß sich zwei Geräte zur gleichen Zeit melden, muß man den Geräten Prioritäten zuordnen.
Dies geschieht mit Hilfe von 5 Requestleitungen:
BR 4, BR 5, BR 6, BR 7, NPR.
Die höchste Priorität ohne Einschränkung besitzen Geräte an der NPR-Leitung (NPR=non processor request). Hier werden Geräte angeschlossen, die direkt mit dem Speicher korrespondieren ohne Mitwirkung des Prozessors (DMA = direct memory access).
Ein Interruptsignal von einem Gerät mit dieser Priorität wird vom Prozessor in jedem Fall akzeptiert.
Alle anderen Requests kommen nur dann zur Geltung, wenn die CPU im PSW eine niedrigere Priorität enthält als die entsprechende Requestleitung.



B. Horizontale Priorität:

An einer Requestleitung können mehrere Geräte liegen. Da aber die Geräte alle "interruptfähig" sind, muß auch hier eine Priorität verteilt werden. Zu diesem Zweck wird die GRANT-Leitung vom Prozessor durch jedes Interface geschleift. Sendet ein Gerät BUS-REQUEST (BR), wird diese Leitung im betreffenden Interface für die folgenden Geräte unterbrochen und das vom Prozessor aktivierte GRANT-Signal hat nur für dieses Gerät Gültigkeit. Durch diese Serienschaltung ergibt sich, daß das Gerät, welches der CPU am nächsten liegt die höhere Priorität besitzt.



3. Interrupt-Ablauf

1. Ist ein Gerät zum Datenaustausch bereit, setzt es ein DONE- bzw. READY-Bit in seinem Statuswort. Wurde das INTERRUPT ENABLE Bit gesetzt, erzeugt das Gerät dadurch einen Bus-Request.

2. Der UNIBUS-Controller der CPU vergleicht die Priorität des Gerätes mit der aktuellen Priorität des Prozessors.

3. Die laufende Instruktion wird fertig abgearbeitet.

4. Ist die Priorität des Prozessors höher, wird der Request ignoriert. Ist aber die Priorität des Requests höher als die aktuelle Priorität des Prozessors, so sendet der Prozessor ein Quittungssignal über die "BUS GRANT"-Leitung und das Gerät erhält die Kontrolle über den Bus.

5. Sobald ein Gerät "BUS MASTER" ist, sendet es zusammen mit dem INTERRUPT-Signal den VEKTOR. Dieser verweist den Prozessor auf das Statuswort und die Startadresse der Interrupt-Serviceroutine.

6. Die in der CPU ablaufende Sequenz ist (bereits im Kap. 4.3.2 gezeigt) bei jeder Art von Unterbrechung gleich:

   1. PS_{alt} ⟶ -(SP)
   2. PC_{alt} ⟶ -(SP)
   3. (Vector) ⟶ PC
   4. (Vector+2) ⟶ PS

7. Dadurch wird die Interrupt-Routine aufgerufen. Der ganze Ablauf bis hierher ist rein hardwaremässig festgelegt; insbesondere gilt: SP = R6!

8. Mit dem Befehl RTI wird die Interrupt-Routine beendet. Der alte PC und PS werden vom Stack geholt und in die CPU geladen.

   RTI

   1. (SP)+ ⟶ PC
   2. (SP)+ ⟶ PS

9. Eine Interrupt-Routine kann jederzeit durch einen Bus-Request, der höhere Priorität besitzt, unterbrochen werden.

   Bei einem solchen Interrupt werden der PC und PS der arbeitenden Interrupt-Routine ebenfalls auf den Stack gerettet, und die neue Interrupt-Routine wird gestartet. Die Verschachtelung von Interrupts kann bis zu einem beliebigen Level gehen (Nesting). Sie wird nur durch einen möglichen Stack-Overflow begrenzt.

4. Regeln für die Interrupt-Programmierung

Wie gezeigt, können Interrupts beliebig verschachtelt werden. Bei der Behandlung der Interrupts und der Festlegung der Prioritäten müssen deshalb die folgenden Regeln beachtet werden:

> 1. Regel: Der Level im Status-Wort der Interrupt-Routine sollte mindestens gleich hoch sein wie der Level der Bus Request Line, an die das entsprechende Gerät angeschlossen ist.

Damit wird verhindert, daß weniger dringende Interrupts (auf tieferem Prioritätsniveau) ein zeitkritisches Interruptprogramm unterbrechen.

> Beispiel: Würde das Interruptprogramm des Kartenlesers (BR 6) vom TTY Interrupt (BR4) unterbrochen,wäre es möglich, daß die Information einer Lochkartenkolonne verlorengeht.

Diese Regel setzt voraus, daß die Hardwareprioritäten vernünftig zugeteilt sind!

> 2. Regel: Die Priorität einer Interrupt-Routine soll erst dann heruntergesetzt werden, wenn der Interrupt behandelt ist.

Die Interrupt-Routine soll jedoch nur das absolut Notwendige erledigen (Datentransfer, Fehlererkennung,...) und alles weniger Dringliche dem Hauptprogramm (resp. einem durch programmierten Interrupt ausgelösten Programm auf tiefere Priorität) überlassen.

> 3. Regel: Sämtliche Register, die in einer Interrupt-Routine gebraucht werden, sind am Anfang zu retten und vor Verlassen der Routine wieder zurückzuladen.

Beim Eintritt in eine Interrupt-Routine sind die Register mit den Werten des unterbrochenen Programmes geladen. Bei der Rückkehr aus der Interrupt-Routine muß dieses Programm die gleichen Werte wie unmittelbar vor der Unterbrechung vorfinden. Man erreicht dies, indem man alle Register, die in der Interrupt-Routine gebraucht werden, am Anfang rettet und am Schluß der Interrupt-Routine wieder zurücklädt.

> 4. Regel: Der gesamte Vektorbereich des Arbeitsspeichers muß sinnvoll definiert sein.

Beispiel: Kapitel 4.5.6

ARBEITSBLATT 4.5

1. Wann wird ein Interrupt-Signal von einem Interface erzeugt?

2. Was ist der Inhalt der Interrupt-Vektor-Adresse?

3. Wie bekommt ein Gerät Bus-Kontrolle?

4. Warum haben NPR's die höchste Priorität?

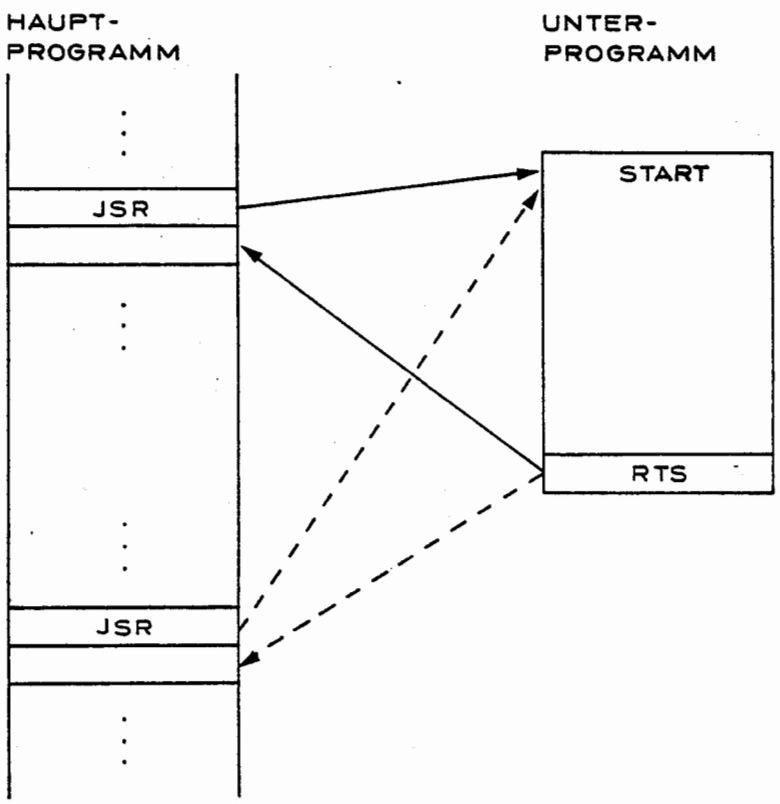
5. Welches Gerät hat die höchste horizontale Priorität?

6. Woraus besteht der Initialisierungs-Teil eines Interrupt-Programms?

4.5 Unterprogrammierung (Subroutines)

4.5.1 Einleitung

Soll in einem (Haupt-) Programm ein bestimmter Abschnitt mehrmals (an verschiedenen Stellen) durchlaufen werden, so kann dieser als Unterprogramm geschrieben werden. An jeder entsprechenden Stelle des Hauptprogrammes steht nun statt des gesamten Abschnittes lediglich ein Sprungbefehl in das Unterprogramm (JSR), am Ende des Unterprogramms ein Rücksprungbefehl (RTS) ins Hauptprogramm.



Anmerkung: Wie bei Interrupts ist auch hier die Möglichkeit des "Nestings" gegeben, d.h. ein Unterprogramm ruft ein weiteres Unterprogramm (eventuell sich selbst) auf. Diese Methode wird bei rekursiven Prozeduren in höheren Programmiersprachen verwendet.

4.5.2 Unterprogrammsprung, JSR

Vor einem Sprung in ein Unterprogramm muß die Rückkehradresse gerettet werden. Diese Adresse steht aber gerade im PC, da dieser bereits auf den nächsten Befehl des Hauptprogrammes zeigt. Der Sprung selbst wird durch das Laden der Zieladresse in den PC ausgelöst. Der eben beschriebene Vorgang wird durch den Befehl JSR, Jump to Subroutine, bewirkt:

JSR: Code 004RDD

1. Zieladresse ⟶ Temp. Reg.
2. Link-Reg. ⟶ -(SP)
3. PC ⟶ Link-Reg.
4. Temp.Reg. ⟶ PC

1. Die Zieladresse wird in einem CPU-internen Register (Temp.Reg.) zwischengespeichert. Von wo diese Adresse geholt wird ergibt sich aus Destination-Mode und Register des JSR-Befehls (DD).

2. Der Inhalt eines der allgemeinen CPU-Register -bei der Unterprogrammierung als "Link-Register" bezeichnet- wird in den Stack abgelegt. Dieses Register wird ebenfalls im JSR-Code angegeben (R).

3. Der Inhalt des PC (=Rückkehr- bzw. Link-Adresse) wird in das Linkregister geladen.

4. Die Zieladresse wird vom Temp.Reg. in den PC gebracht. Damit wird das Unterprogramm gestartet.

4.5.3 Unterprogrammrücksprung, RTS

Die Rückkehr aus dem Unterprogramm wird durch den Befehl RTS, Return from Subroutine, ausgeführt. Da die Rückkehradresse vom JSR-Befehl ins Link-Register gerettet wurde, braucht der RTS-Befehl nur auf dieses zuzugreifen:

RTS: Code 00020R

1. Link-Reg. ⟶ PC
2. (SP)+ ⟶ Link-Reg.

1. Der Inhalt des im RTS-Befehl angegebenen Link-Registers wird in den PC geladen. Die Angabe des Link-Registers muß in JSR und RTS identisch sein (Ausnahmen nur durch besondere programmtechnische Vorkehrungen möglich)!

2. Der in den Stack gerettete Inhalt des zum "Linken" benutzten CPU-Registers wird zurückgeladen.

ARBEITSBLATT 4.6

Beantworten Sie bitte folgende Fragen:

1. Was ist der Inhalt des SP nach Ausführung dieser Befehle?

```
MOV #500,SP

MOV R0,-(SP)

MOV R1,-(SP)
```

2. Wie ist der interne Ablauf des Befehls JSR R5,AUS ?

3. Wie ist der interne Ablauf des Befehls RTS PC ?

4. Wie rufen Sie die Unterroutine PRINT auf?

```
                    .
                    .
                    .
          ----------------------
                    .
                    .
                    .
                    .
                    .
                    .
PRINT: TSTB @#TKS
                    .
                    .
                    .
       RTS R4
```

4.5.4 Übertragung von Parametern in Unterprogramme (Linkage)

Soll ein Unterprogramm mit Argumenten (Operanden) versorgt werden, so geschieht dies mit Hilfe des Link-Registers. Das Linkregister beinhaltet, wie bereits erläutert, die Adresse der auf den JSR-Befehl folgenden Speicherzelle. Diese Speicherzelle beinhaltet entweder

- den nächsten Befehl des Hauptprogramms, oder
- einen Operanden, oder
- die Adresse eines Operanden

Den letzten beiden Möglichkeiten ist dieses Kapitel gewidmet. Ist z.B. R5 das Link-Register, kann das erste Argument über die Adressierungsarten (R5), (R5)+, X(R5) für Operanden oder über (R5)+ usw. für die Adresse der Operanden verwendet werden. Wird der Auto-Increment-Mode verwendet, so wird das Link-Register automatisch richtig gesetzt und zeigt zum nächsten Argument oder -falls die Operanden/Adressen nach dem JSR-Befehl im Hauptprogramm stehen- eventuell auf den nächsten Befehl des Hauptprogramms.

<u>1. Beispiel:</u>

```
10400   JSR R5,SUBR          ;Sprung in das
10402   <SUBR>               ;Unterprogramm SUBR
10404   <ARGUMENT Nr.1>      ;Argumentenliste für
10406   <    "    Nr.2>      ;Unterprogramm
10410   <"nächster Befehl">
```

```
20306   SUBR: MOV (R5)+,R1   ;Zugriff auf Argumentliste
              MOV (R5)+,R2   ;mit Hilfe von R5
                  .          ;R5 zeigt jetzt auf den
                  .          ;nächsten Befehl im
                  .          ;Hauptprogramm
              RTS            ;und der RTS-Befehl wird
                             ;korrekt ausgeführt.
```

2. Beispiel:

```
10400   JSR R5,SUBR          ;Sprung in das
10402   <SUBR>               ;Unterprogramm SUBR
10404   77722                ;Adresse des 1.Arguments
10406   50304                ;   "      "  2.      "
10410   30002                ;   "      "  3.      "
10412   <nächster Befehl>
   .
   .
   .

20306   SUBR:MOV@(R5)+,R1    ;Zugriff (indirekt) auf
20300        MOV@(R5)+,R2    ;die Argumente
                  .          ;R5 zeigt auf den nächsten
                  .          ;Befehl im Hauptprogramm
             RTS             ;Rückkehr ins    "    "
```

Anmerkung: Sind die Argumente alle in einer Liste mit aufsteigenden Adressen abgelegt, vereinfacht sich das Beispiel, da in 10404 nur die Startadresse der Liste angegeben werden muß.
Zusätzlicher Vorteil der Angabe von Adressen von Operanden (-Listen): Unterprogramme und Operanden können beliebig im Speicher verstreut sein. Das Hauptprogramm muß nicht die Opranden beinhalten.

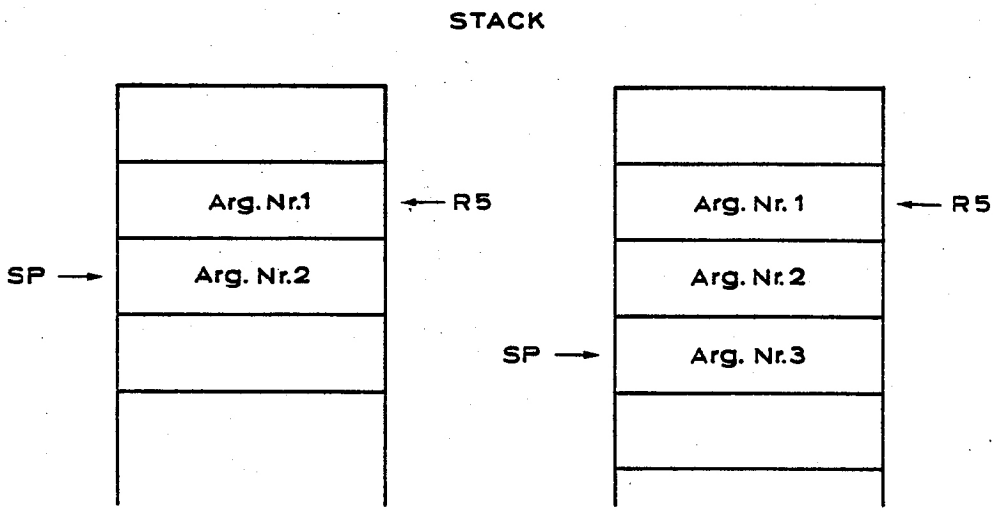
3. Beispiel:

In einem Unterprogramm sollen 2 Worte einer Operanden-Liste addiert werden. Das Ergebnis soll in der Liste stehen (anstelle des 2. Operanden).

```
MOV LISTADR,R1              ;Listenanfangsadresse nach R1
                            ;laden
JSR PC,SUBR                 ;PC als Linkregister, wird
                            ;immer angegeben wenn das
                            ;Unterprogramm kein
                            ;Linkregister benötigt.
SUBR: ADD (R1)+,(R1)        ;Addiere 1. Operanden auf 2.
                            ;Operanden
                            ;R1 zeigt auf 2. Operanden
                            ;(-Ergebnis)
      oder
      ADD (R1),2(R1)        ;wie oben, jedoch zeigt R1 nach
                            ;wie vor auf den 1. Operanden
```

4. Beispiel:

Da es grundsätzlich möglich ist, Argumente vom Stack zu holen, ist es empfehlenswert eine Kopie von R6 (=Stackpointer) in ein anderes Register abzuspeichern. In folgendem Beispiel wurde R6 vor Abspeichern der Argumente nach R5 gerettet. Somit zeigt R5 stets auf den Anfang der Stacktabelle, unabhängig von der Anzahl der Argumente:



Argument Nr. 2 steht in 2(R5)

Argument Nr. 2 steht ebenfalls in 2(R5)

Würde man den Stackpointer als Argumentzeiger verwenden:

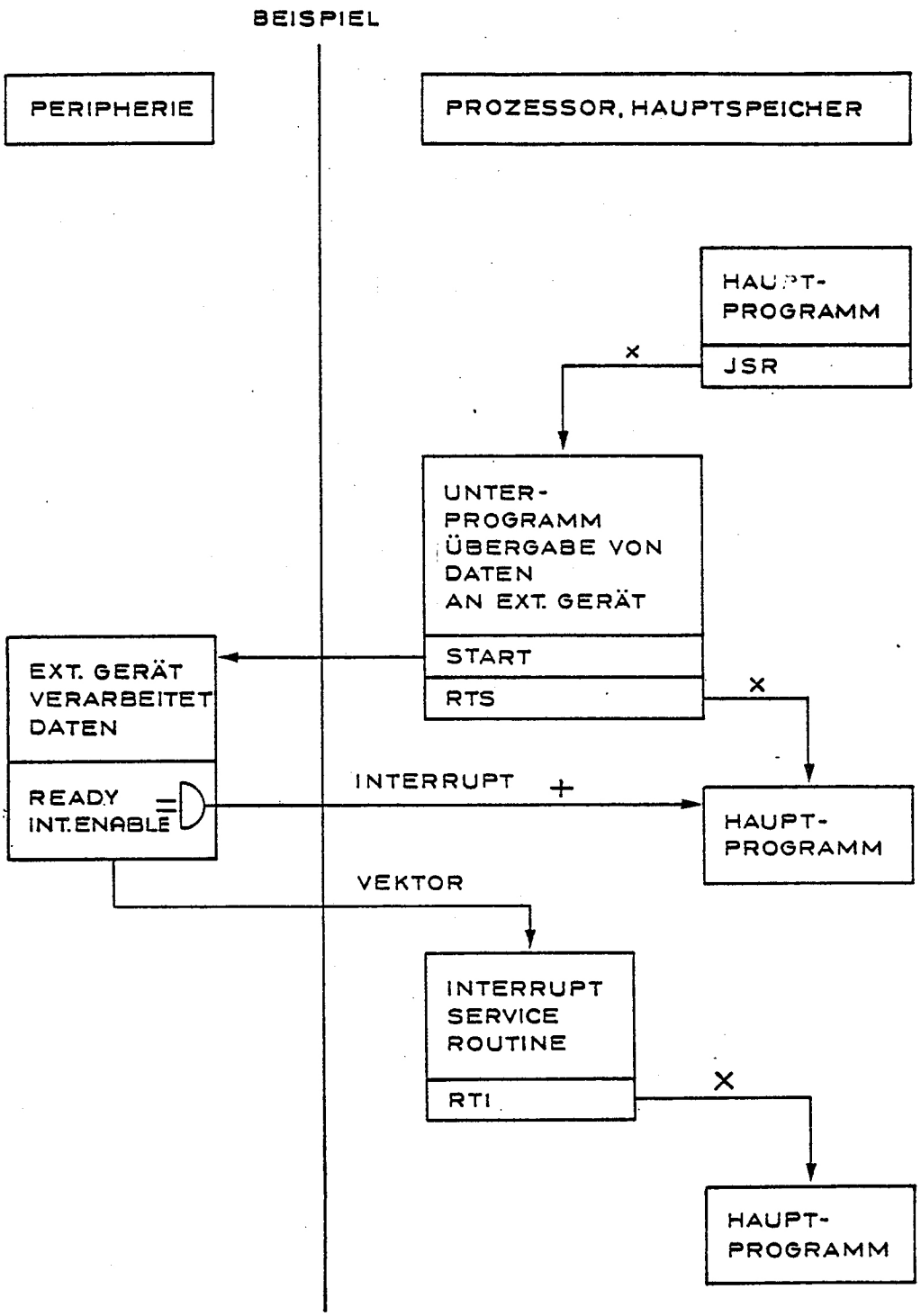
Argument Nr. 2 steht in (SP)

Argument Nr. 2 steht in -2(SP)

Man müßte also ständig über die Anzahl der Argumente bzw. den Stand des SP informiert sein.

Ausführliche Beispiele siehe Anhang A.2

4.5.5 Vergleich: Interrupt - Unterprogramm (Beispiel)



x PROGRAMMGESTEUERTER ÜBERGANG
\+ EREIGNISGESTEUERTER ÜBERGANG

4.5.6 E/A-Programmierung mit Interrupt

Die E/A-Programmbeispiele der vorangegangenen Kapitel haben den großen Nachteil, daß der Rechner wertvolle Zeit in einer Warteschleife vergeudet.
Wie in Kapitel 4.3.3 erläutert, schafft die Möglichkeit der Programmunterbrechnung (Interrupt) eine sehr effiziente Abhilfe: Nach Übergabe/Übernahme eines Zeichens beginnt der Prozessor eine andere Tätigkeit und das Interface meldet sich per Interrupt wenn es ein neues Zeichen übernehmen/übergeben kann. Der Prozessor unterbricht seine Tätigkeit nur für die Zeit der Übergabe/Übernahme des nächsten Zeichens.
Alle vorbereitenden Tätigkeiten die für einen solchen Vorgang notwendig sind, werden an folgendem Beispiel erläutert.

Beispiel: E/A-Programm mit Interrupt

Ein Programm welches fortlaufend ASCII-Zeichen ausdruckt, soll durch das Drücken einer beliebigen Taste der Terminaltastatur unterbrochen werden.
Das gedrückte Zeichen soll auf dem Drucker erscheinen (Echo) und anschließend der ASCII-Ausdruck fortgesetzt werden.
Lesen Sie dieses Programm aufmerksam und kommentieren Sie es!

```
1000    Start:   MOV #1000,SP
1004             CLR @#177776
1010             MOV #1050,@#60
1016             CLR @#62
1022             MOV #100,@#177560
1030    ASCII:   TSTB @#177564
1034             BPL ASCII
1036             MOV RØ,  177566
1042             INC RØ
1044             JMP @#ASCII
1050    INT:     TSTB @#177564
1054             BPL INT
1056             MOV @#177562,@#177566
1064             RTI
```

ANHANG

A.1 Bedienungsanleitungen

A.1.1 LSI 11-ODT

A.1.2 PDP 11/04, 11/34 Console Emulator

A.1.3 PDP 11/04, 11/34 Programmers Console

A.1.4 PDP 11/05, 11/10 Switch Panel

A.1.5 PDP 11/35, 11/40 Switch Panel

A.1.6 PDP 11/44 Console

A.2 Programmbeispiele

A.3 Abkürzungen

A.1 Bedienungsanleitungen

A.1.1 LSI-ODT (On-line Debugging Tool)

Für die Fehlersuche bei Programmen hat Digital Equipment das Testprogramm ODT entwickelt, das entweder von einem lokalen oder entfernten Terminal bedient werden kann. Dieses Programm ist ein Teil des Processor-Microcodes und ermöglicht der Zentraleinheit, auf Befehle und Informationen zu reagieren, die durch das Terminal eingegeben werden. Die Kommunikation selbst ist eine Folge von ASCII-Zeichen, die von der Zentraleinheit als Konsolbefehle interpretiert werden.

Die ODT-Kommandos kann man immer benutzen, wenn der Prozessor im HALT-Mode ist.

In den HALT-Mode kommt man:

1. durch Drücken der BREAK-Taste (Funktion kann unterdrückt werden);

2. bei der Ausführung einer HALT-Instruktion;

3. wenn der Rechner bestimmte Fehler erkennt;

4. wenn man die HALT-Taste betätigt.

Hält der Prozessor das laufende Programm an, dann erfolgt folgender Ausdruck:

```
xxxxxx
@
```

Die Zahl xxxxxx ist die oktale Adresse des nächsten Befehles und @ ist der Prompt-Charakter des ODT.

ODT-Kommandos

n/ Der Inhalt der Speicherzelle n (oktal) wird ausgedruckt. Dahinter kann ein neuer Wert angegeben werden (mit Abschlußzeichen).

<CR> Abschlußzeichen für angesprochene Zelle (Speicher Register)

<LF> Abschlußzeichen für angesprochene Zelle (Speicher Register) und Ausdruck der nächst höheren Zelle (Speicher,Register)

%n/ oder

Rn/ Der Inhalt des Registers n (Ø-7) wird ausgedruckt. Dahinter kann ein neuer Wert angegeben werden (mit Abschlußzeichen)

RS/ Der Inhalt des PSW wird ausgedruckt

nG Ein Programm wird bei Adresse n gestartet

P Ein Programm wird fortgesetzt (nach HALT-Befehl)

Single Step wird ein Programm bei eingeschalteter HALT-Taste mit nG bei Adresse n gestartet, so kann es mit P im Einzelschritt abgearbeitet werden.

Anmerkung: Das ODT bei der 11/23 arbeitet mit 18-Bit-Adressen, auch wenn keine Memory-Management-Einheit vorhanden ist oder diese nicht eingeschaltet wurde.

A.1.2 PDP 11/04, 11/34 Console Emulator

Der Console Emulator ist ein in ROM's abgelegtes Hilfsprogramm welches die Eingabe von Programmen über eine Terminaltastatur ermöglicht. Im Normalfall wird dieses Programm beim Einschalten der Maschine oder durch Betätigen der BOOT-Taste (bzw. CNTR-BOOT) gestartet. Das Programm meldet sich nach dem Ausdruck der CPU Register

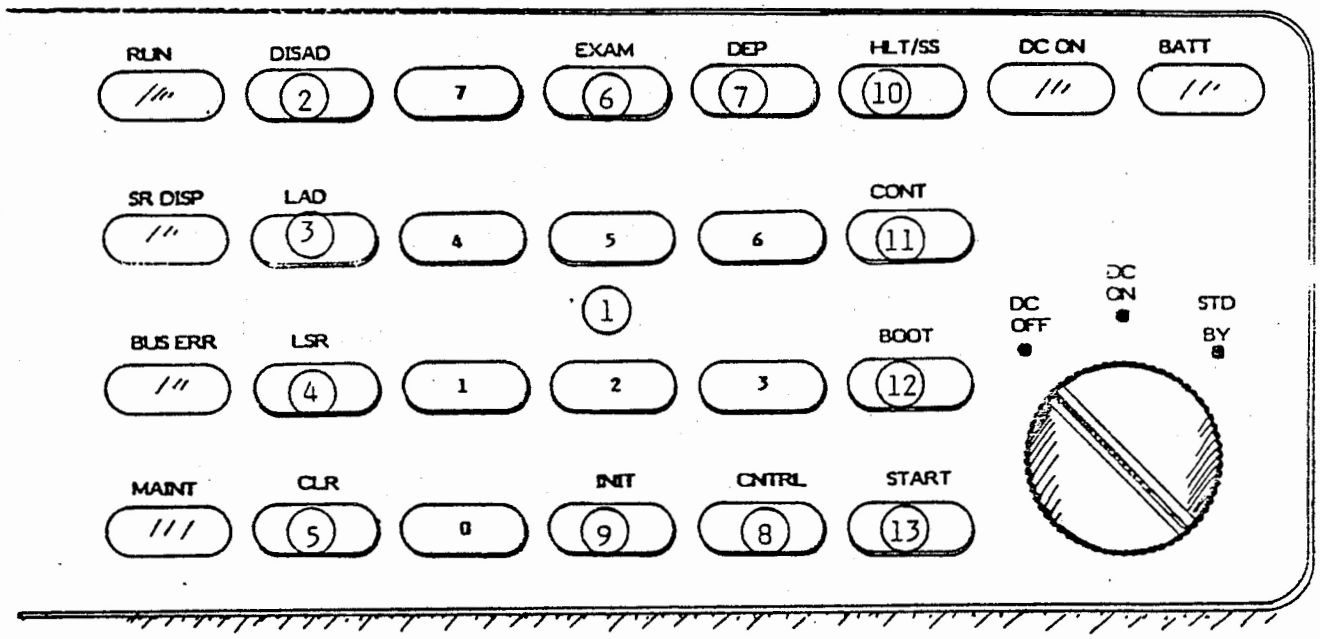
R0 R4 R6 R5("OLD PC") mit dem Prompt-Zeichen

$ oder @

Nun können folgende Kommandos eingegeben werden:

@ L xxxxxx <CR>	Laden der Adresse xxxxxx (oktal!).
@ E YYYYYY	Examine: Ausdruck des Inhalts der vorher eingegebenen Adresse.
@ D ZZZZZZ <CR>	Eingabe des Werts ZZZZZZ (oktal) in die vorher eingegebene Adresse.
@ S <CR>	Start des Programms. Startadresse vorher eingeben.
@ DK0	Start eines Bootprogramms (hier für RK05-Platte). Dieses Programm holt den Monitor von externen Speichergeräten (Platte, Band) und startet diesen.

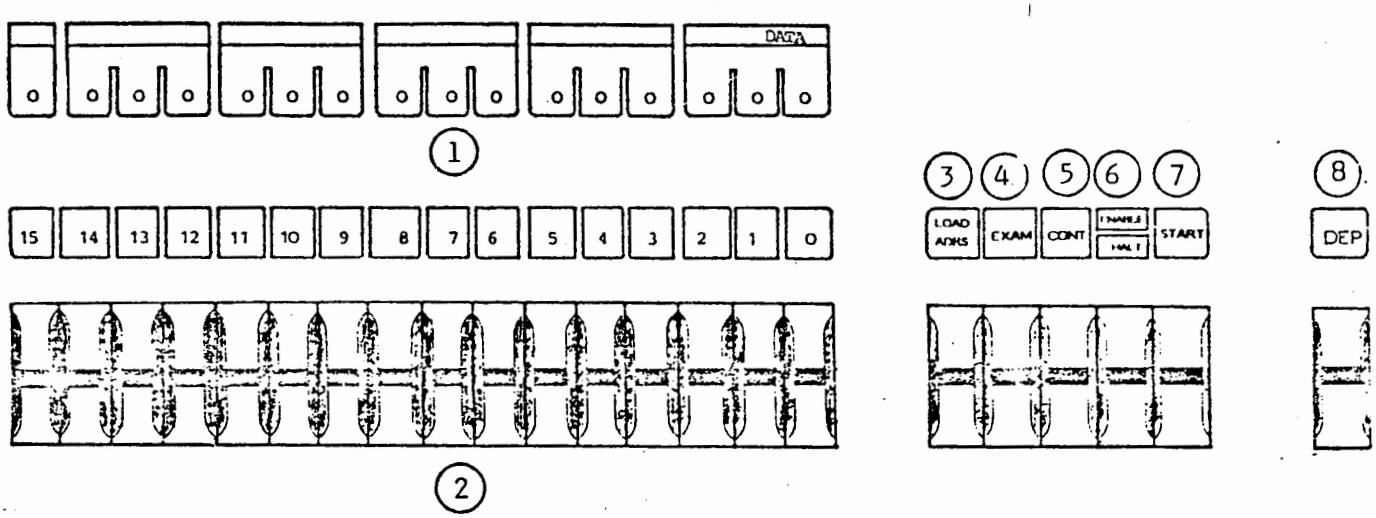
A.1.3 PDP 11/04, 11/34 Programmers Console



1	OKTALE TASTEN	Oktale Zahlen werden eingegeben.
2	DISPLAY ADDRESS	Die laufende Adresse wird angezeigt.
3	LOAD ADDRESS	Der Inhalt des Display-Registers wird in den Prozessor gebracht.
4	LOAD SWITCH REGISTER	Der Inhalt des Display-Registers wird in die Adresse 177570 gebracht.
5	CLEAR	Das Display wird gelöscht.

6	EXAMINE	Der Inhalt der geladenen Adresse wird angezeigt. Ein wiederholtes Drücken zeigt den Inhalt der nächsten Adresse an.
7	DEPOSIT	Der Inhalt des Display-Registers wird in die geladene Adresse gebracht.
8	CNTRL (CONTROL)	Diese Taste wird in Verbindung mit anderen Tasten verwendet.
9	INITIALIZE (mit CNTRL)	
10	HALT/SINGLE STEP(mit CNTRL)	Der Prozessor wird angehalten.
	HALT/SINGLE STEP (ohne CNTRL)	Ermöglicht Single-Step-Mode.
11	CONTINUE (mit CNTRL)	Nach einem HALT wird die Abarbeitung des Programms forgesetzt.
12	BOOT (mit CNTRL)	Das Bootstrap-Programm wird gestartet.
13	START (mit CNTRL)	Der Prozessor wird an der geladenen Adresse gestartet.

A.1.4 PDP 11/05, 11/10 Switch Panel



1	ADDRESS/DATA DISPLAY	Diese 16 Lämpchen (16 Bits) zeigen entweder Daten oder Adressen, abhängig von den gedrückten Schaltern, an.
2	SWITCH REGISTER	Binäre Zahlen oder Adressen werden in den Prozessor geladen. Stellung der Schalter oben = 1, unten = 0.
3	LOAD ADDRESS	Der Inhalt des Switch-Registers (eine Adresse) wird in den Prozessor gebracht.
4	EXAMINE	Der Inhalt der geladenen Adresse wird angezeigt. Ein wiederholtes Drücken zeigt den Inhalt der nächsten Adresse.
5	CONTINUE	Der Prozessor läuft weiter, nachdem er von einem HALT Befehl gestoppt wurde.
6	ENABLE/HALT	In Enable-Stellung kann der Prozessor normal arbeiten. In Halt-Position wird der Rechner nach Ausführung der laufenden Instruktionen angehalten.
7	START	Der Prozessor wird an der geladenen Adresse gestartet.
8	DEPOSIT	Der Inhalt des Switch-Registers wird in die geladene Adresse gespeichert. Durch wiederholtes Anheben wird der Inhalt des Switch-Registers in die folgenden Adressen gebracht.

A.1.5 PDP 11/35, 11/40 Switch Panel

1	ADRESS REGISTER DISPLAY	Anzeige der 18-Bit-Adressen bei LOAD ADRS, DEP u. EXAM. Bei HALT u. WAIT wird die nachfolgende Adresse angezeigt.
	DATA REGISTER DISPLAY	Anzeige der 16 Bit Daten bei EXAM, DEP. Bei Single Step wird PSW angezeigt. Bei HALT u. RESET wird RØ angezeigt. Bei WAIT wird Instruktions Register angezeigt.
2	SWITCH REGISTER	Schalterfeld zur Eingabe von Daten (16 Bit) u. Adressen (18 Bit) per DEP bzw. LOAD ADRS.
3	LOAD ADRS	Der Switch Register-Inhalt wird als Adresse in die CPU eingegeben.
4	EXAM	Der Inhalt einer (vorher) geladenen Adresse wird auf dem Data Register Display angezeigt. Wiederholtes Drücken zeigt den Inhalt der nächsten Adresse.
5	CONT	Der Prozessor läuft weiter nachdem er durch einen HALT- Befehl gestoppt wurde. Bei gedrückter HALT-Taste kann ein Programm mit CONT schrittweise abgearbeitet werden (Single Step).
6	ENABLE/HALT	HALT: Anhalten eines laufenden Programms bzw. für Single Step (mit CONT). ENABLE: Freigabe für START eines Programms.

7	START	Das Programm wird an der vorher eingegebenen Adresse (LOAD ADRS) gestartet (mit System-Reset).
8	DEP	Der Inhalt des Switch Registers wird in die vorher geladene Adresse (LOAD ADRS) abgespeichert (Deposit). Die Adresse wird automatisch erhöht.
9	Status Lämpchen	
	RUN	leuchtet: Prozessor-Clock läuft, HALT, WAIT gelöscht: Prozessor erhält Daten von Peripherie bzw. bei RESET
	PROC	leuchtet: Prozessor ist BUS-MASTER
	BUS	leuchtet: UNIBUS wird benutzt
	CONSOLE	leuchtet: Console Mode, Prozessor steht
	USER	leuchtet: Befehle im USER-MODE (Memory Management) werden abgearbeitet
	VIRTUAL	leuchtet: Adress Register Display zeigt eine virtuelle 16 Bit Adresse an (Mem. Management)

Schlüsselschalter

OFF/POWER/PANEL LOCK

In PANEL LOCK-Stellung sind die CONTROL SWITCHES außer Betrieb.

Anmerkungen

Ungerade Adressen bei EXAM liefern das ganze Datenwort (even address). Bei Verwendung nicht existierender Adressen wird bei EXAM der Inhalt des Switch Register angezeigt. Bei DEP kommt die Adresse des Switch Registers im Data Display zur Anzeige (177570).

A.1.6 Bedienungsanleitung 11/44

1. Einschalten oder CNTRL P u. H <CR> oder BOOT-Taste
   ergibt:
   ```
   CONSOLE
   17777707 xxxxxx
   >>>
   ```

2. Ansprechen von Speicherzellen

>>> D 1000 12345	1000 mit 12345 laden
>>> E 1000 od. E *	Inhalt von 1000 prüfen
D 1000 1111 D + 2222 D + 3333	Aufeinanderfolgende Zellen laden
E 1000 E E	Inhalt aufeinanderfolgender Zellen prüfen
D * 4444	Die zuletzt angesprochene Zelle ändern/laden und
E *	prüfen

3. Ansprechen von Registern der CPU

D/G 7 1000	Register 7 (PC) mit 1000
E/G 7	laden und prüfen

4. Speicherbereich überprüfen

E/N:12 1000	12_8 Speicherzellen überprüfen, beginnend bei 1000.

5. CPU-Register überprüfen

E/G/N:10 0	Reg. Ø...7

6. Programm starten

   ```
   S 1000
   ```

7. Single Step
   Schalter: HALT
   ```
   S 1000
   C
   ```

A.2 Programmbeispiele

1. Programmbeispiel:

Dieses Programm soll den Inhalt von 30_8 Byte-Adressen löschen und anschließend anhalten.

R0 = 600
R1 = 30

```
A:  CLRB (RØ)+      105020
    DEC R1          005301
    BNE A           001375
    HALT            000000
```

R1 enthält die Anzahl der Adressen, die gelöscht werden sollen. R1 ist also ein Zähler. Das Abzählen geschieht mit DEC R1, d.h. jedesmal, wenn eine Byteadresse gelöscht ist, wird der Inhalt von R1 decrementiert.
BNE testet die Conditioncodes im PSW und führt die Verzweigung solange aus, bis das Z-Bit im PSW 1 ist, d.h. bis das Register 1 den Inhalt 0 hat und erreicht dann den HALT-Befehl.

2. Programmbeispiel:

Multiplikation:

Eine Binärzahl wird mit 2 multipliziert, indem man die Zahl um eine Binärstelle nach links verschiebt. Diese Verschiebung führt der Befehl ASL aus.

Aufgabe: $5_{10} \times 4_{10}$

Ein Register wird mit ($4_{10} = 2^2$) 2 geladen und ein anderes mit der zu multiplizierenden Zahl.

```
R1 = 5
R2 = 2


START:  ASL R1
        DEC R2
        BNE START
        HALT
```

Das Ergebnis in R1 = $10100_2 = 20_{10}$

Division:
Bei der Division wird genau so verfahren, nur wird statt ASL der Befehl ASR (arithmetic shift right) verwendet.

3. Programmbeispiel:

Ein Speicherblock von Adresse 1000 bis Adresse 1500 (incl.) soll nach Adresse 3000 verlegt werden.

```
     MOV #1000,R2
     MOV #3000,R3
A:   MOV (R2)+,(R3)+
     CMP R2,#1502
     BNE A
     HALT
```

R2 zeigt auf das jeweils nächste Wort, welches in die durch R3 adressierte Zelle geladen wird. Nach jedem Transfer wird abgefragt, ob der letzte Speicherinhalt schon übertragen wurde.

4. Programmbeispiel:

Memory Test: Dieses Programm löscht zunächst jede Speicherzelle und schreibt dann 1...1 in die Zelle und liest diesen Wert wieder aus. Anschließend wird Ø...Ø geschrieben und ausgelesen bzw. verglichen.

```
 0      012700          MOV #TOPADDRESS,R0
 2      TOPADDRESS
 4      012701          MOV #LOW ADDRESS,R1
 6      LOW ADDRESS
10      005040      A:  CLR -(R0)
12      005110          COM (R0)
14      022710          CMP #177777,(R0)
16      177777
20      001005          BNE B
22      005110          COM (R0)
24      001003          BNE B
26      020001          CMP R0,R1
30      001365          BNE A
32      000000          HALT
34      000000      B:  HALT
```

In RØ und R1 wird der Speicherbereich vorgegeben. Abhängig vom Ergebnis des Tests endet das Programm in Zelle 32 (Test erfolgreich) oder Zelle 34 (Memory-Fehler).
Im Fehlerfall steht die Fehleradresse in RØ.

5. Beispiele zur Ein-/Ausgabe-Programmierung

A) Abspeichern von Text ab Zelle 5000, bis "!" gedrückt wird. Gemäß Flußdiagramm Kap. 4.5.3

```
   MOV #5000,RØ          ;Adresse des 1. Zeichens
A: TSTB @#177560         ;DONE BIT testen
   BPL A                 ;Falls gesetzt: Progr.
                          fortsetzen
   MOVB @#177562,(RØ)    ;Abspeichern des Zeichens
   BICB #200,(RØ)        ;Parity Bit löschen
   MOVB @#177562,@#177566 ;Echo
   CMPB #41,(RØ)+        ;war abgespeichertes
                          Zeichen ein "!"?
   BNE A                 ;Falls Nein: Nächstes
                          Zeichen abspeichern
   HALT                  ;Falls Ja: HALT
```

B) Ausgabe von Text. Gemäß Flußdiagramm Kap.4.5.4

```
   MOV #5000,RØ          ;Adresse des 1. Zeichens
A: TSTB @#177564         ;READY BIT testen
   BPL A                 ;Falls gesetzt: Progr.
                          fortsetzen
   MOVB (RØ),@#1777566   ;Ausgabe eines Zeichens
   CMPB (RØ)+,# 41       ;war gedrucktes Zeichen
                          ein "!"?
   BNE A                 ;Falls Nein: Nächstes
                          Zeichen drucken
   HALT                  ;Falls Ja: HALT
```

6. Beispiele zur Unterprogrammierung

A) Dieses Programm addiert zwei Zahlen in der Unterroutine die in den Speicherzellen ZAHL 1 und ZAHL 2 abgelegt sind. Das Ergebnis wird in die Zelle SUM geladen.

```
000500  012706  START:  MOV #500,SP
        000500
000504  012767           MOV #4,ZAHL1
        000004
        000036
000512  012767           MOV #5,ZAHL2
        000005
        000032
000520  004767           JSR PC,ADD
        000006
000524  016704           MOV SUM,R4
        000024
000530  000000           HALT
000532  016700  ADD:     MOV ZAHL1,R0
        000012
000536  066700           ADD ZAHL2,R0
        000010
000542  010067           MOV R0,SUM
        000006
000546  000207           RTS PC
000550  000000  ZAHL1:   Ø
000552  000000  ZAHL2:   Ø
000554  000000  SUM:     Ø
```

B) Dieses Programm addiert zwei Zahlen in der Unterroutine die hinter dem JSR-Befehl abgelegt sind.

```
000500  012706  START:  MOV #500,SP
        000500
000504  004567           JSR R5,ADD
        000006
000510  000004           4
000512  000005           5
000514  000000           HALT

000516  012500  ADD:     MOV (R5)+,RØ
000520  062500           ADD (R5)+,RØ
000522  000205           RTS R5
```

C) Dieses Programm addiert zwei Zahlen in der Unterroutine deren Adressen in einer Adresstabelle gespeichert sind.

```
000500  012706          MOV #500,SP
        000500
000504  012767          MOV #4,FELD1
        000004
        000044
000512  012767          MOV #5,FELD2
        000005
        000040
000520  012767          MOV #2,TAB
        000002
        000022
000526  012705          MOV #TAB,R5
        000550
000532  004767          JSR PC,ADD
        000002
000536  000000          HALT
000540  012500  ADD:    MOV (R5)+,R0
000542  013504          MOV @(R5)+,R4
000544  063504          ADD @(R5)+,R4
000546  000207          RTS PC
000550  000000  TAB:    0
000552  000556          556
000554  000560          560
000556  000000  FELD1:  0
000560  000000  FELD2:  0
```

D) Dieses Programm addiert zwei Zahlen in der Unterroutine die auf dem Stack abgelegt sind.

```
000500  012706  START:  MOV #500,SP
        000500
000504  010146          MOV R1,-(SP)
000506  012746          MOV #4,-(SP)
        000004
000512  012746          MOV #5,-(SP)
        000005
000516  004767          JSR PC,ADD
        000002
000522  000000          HALT
000524  012601  ADD:    MOV (SP)+,R1
000526  012600          MOV (SP)+,R0
000530  062600          ADD (SP)+,R0
000532  000201          RTS R1
```

A.3 Abkürzungen

ABS	absolute
A/D	analog-to-digital
ADC	add carry
ADRS	address
ASCII	American Standard Code for Information Interchange
ASL	arithmetic shift left
ASR	arithmetic shift right
	automatic send/receive
B	byte
BAR	bus address register
BBSY	bus busy
BCC	branch if carry clear
BCS	branch if carry set
BEQ	branch if equal
BG	bus grant
BGE	branch if greater or equal
BGT	branch if greater than
BHI	branch if higher
BHIS	branch if higher or same
BIC	bit clear
BIS	bit set
BIT	bit test
BLE	branch if less or equal
BLOS	branch if lower or same
BLT	branch if less than
BMI	branch if minus
BNE	branch if not equal
BPL	branch if plus
BR	branch, bus request
BRD	bus register data
BSP	bach space
BSR	bus shift register
	back space record
BSY	busy
BVC	branch if overflow clear
BVS	branch if overflow set

CBR	console bus request
CLC	clear carry
CLK	clock
CLN	clear negative
CLR	clear
CLV	clear overflow
CLZ	clear zero
CMP	compare
CNPR	console non-processor request
CNTL	control
COM	complement
COND	condition
CONS	console
CONT	contents
	continue
CP	central processor
CSR	control and status register
D	data
D/A	digital-to-analog
DAR	device address register
DATI	data in
DATIP	data in pause
DATO	data out
DATOB	data out, byte
DB	data buffer register
DCDR	decoder
DE	destination effective address
DEC	decrement
	Digital Equipment Corporation
DEL	delay
DEP	deposit
DEPF	deposit flag
DIV	divide
DMA	direct memory access
DSEL	device select
DST	destination
DSX	display, X-deflection register
EAE	extended arithmetic element
EMT	emulator trap
ENB	enable
EOF	end-of-file
EOM	end-of-medium
ERR	error
EX	external
EXAM	examine
EXAMF	examine flag
EXEC	execute
EXR	external reset

F	flat (part of signal name)
FCTN	function
FILO	first in, last out
FLG	flag
GEN	generator
IDIVR	integer divide routine
INC	increment
	increase
INCF	increment flag
IND	indicator
INH	inhibit
INIT	initialize
INST	instruction
INTR	interrupt
INTRF	interrupt flag
I/O	input/output
IOT	input/output trap
IOX	input/output executive routine
IR	instruction register
IRD	instruction register decoder
ISR	instruction shift register
JMP	jump
JSR	jump to subroutine
LIFO	last in, first out
LKS	line time clock status register
LOC	location
LP	line printer
LSB	least-significant bit
LSBY	least-significant byte
LSD	least-significant digit
LTC	line time clock
MA	memory address
MAR	memory address register
MBR	memory buffer register
MEM	memory
ML	memory location
MOV	move
MSB	most-significant bit
MSBY	most-significant byte
MSD	most-significant digit
MSEL	memory select
MSYN	master sync

ND	negative driver
NEG	negative
NOR	normalize
NPG	non-processor grant
NPR	non-processor request
NPRF	non-processor request flag
NS	negative switch
ODT	octal debugging technique
OP	operate
	operation
OPR	operator
	operand
PA	parity available
PAL	program assembly language
PB	parity bit
PC	program counter
PD	positive driver
PDP	programmed data processor
PERIF	peripheral
PGM	program
PP	paper tape punch
PPB	paper tape punch buffer register
PPS	paper tape punch status register
PR	paper tape reader
PRB	paper tape reader buffer register
PROC	processor
PRS	paper tape reader status register
PS	processor status
	positive switch
PTR	priority transfer
PTS	paper tape software system
PUN	punch
RD	read
RDR	reader
REG	register
REL	release
RES	reset
ROL	rotate left
ROM	read-only memory
ROR	rotate right
R/S	rotate/shift
RTI	return from interrupt
RTS	return from subroutine
R/W	read/write
R/WSR	read/write shift register

S	single
SACK	selection acknowledge
SBC	subtract carry
SC	single cycle
SE	source effective address
SEC	set carry
SEL	select
SEN	set negative
SEV	set overflow
SEX	sign extend
SEZ	set zero
SI	single instruction
SP	stack pointer
	spare
SR	switch register
SRC	source
SSYN	slave sync
ST	start
STPM	set trap marker
STR	strobe
SUB	subtract
SVC	service
SWAB	swap byte
TA	trap address
	track address
TEMP	temporary
TDR	timing, driver
TK	teletype keyboard
TKB	teletype keyboard buffer register
TKS	teletype keyboard status register
TP	teletype printer
TPB	teletype printer buffer
TPS	teletype printer status register
TRT	trace trap
TSC	timing state control
TSS	timing, selection switch
TST	test
UTR	user trap
VEC	vector
WC	word count
WCR	word count register
XDR	X-line driver
XRCG	X-line read control group
XWCG	X-line write control group
YDR	Y-line driver
YRCG	Y-line read control group
YWCG	Y-line write control group