GENERAL INSTRUMENT
MICROELECTRONICS

Series 8000 Microprocessor

ADVANCED LOGIC PROCESSOR SYSTEM (ALPS)

— DATA SHEET —

— HARDWARE MANUAL —

— SOFTWARE MANUAL —

PRICE £5.00
DM25.00
$10.00
uivalent

MAY 1976

GENERAL INSTRUMENT CORPORATION • MICROELECTRONICS

GENERAL INSTRUMENT
MICROELECTRONICS

FC 9647

EUROPE
AGENCIES &
DISTRIBUTORS

AUSTRIA
Wien
Elbatex GmbH
Breitenfurterstr. 381
A 1235 Wien
Tel: 86.91.58 Telex: 13128

BELGIUM
C.P. Clare International N.V.
102 Gen. Gratry
Bruxelles 4.
Tel. 02-736.01.97., Telex: 24157

DENMARK
A/S Nordisk-Elektronik
Transformervej 17
DK-2730 Herlev
Tel. 84.20.00, Telex: 19219

FINLAND
Jorma Sarkkinen Ky.
Heikintori, P.O. Box 19.
SF-02100 Tapiola
Tel: 46.10.88, Telex: 122028

FRANCE
P.E.P.
4 Rue Barthelemy
92120 Montrouge
Tel: 735.33.20. Telex: 20453

GERMANY
Lehrte
Altron
A.E. Thronicke KG.
3160 Lehrte
Postfach 1280
Tel: 05132/53024, Telex: 922383
Frankfurt/Main
Berger Elektronik GmbH
Am Tiergarten 14
Tel: 0611/490311, Telex: 04-12649
Heilbronn
Elbatex GmbH
Caecilienstr. 24
Tel: 07131/89001, Telex: 728362
Munchen
Electronic 2000 Vertriebs-GmbH
Neumarkter Str. 75
8000 Munchen 80
Tel: 0 89/43 40 61, Telex: 02-2561

GREECE
Elkon Ltd
46 Aakilpou Str.
Athens
Tel: (021) 529 385, Telex: 214150

HOLLAND
Curijn Hasselaar
V Limburg Stirumstraat 31
P.O. Box 37, Geldermalsen
Tel: (0) 3455-2150, Telex: 40259

IRELAND
Neltronic Ltd
John F. Kennedy Road
Naas Road
Dublin 12
Tel: 501845 Telex: 4837

ITALY
Bologna
Adelsy Sas
Via Savigno 5
Tel: 470622
Milano
Adelsy
Via Jacopo Palma 1
Tel: 4044046-7-8
Roma
Augusto Cioccolanti
Via C. Pascucci 28
00124 Roma
Tel: 60.91.952
Adelsy
Piazzale Flaminio 19
Roma
Tel: 3606580/3605769

NORWAY
J.M. Feiring A/S
Box 101, Bryn, Oslo 6
Tel: (02) 68.63.60, Telex: 16435

SPAIN
Orbetronik S.A.
Garduxer 16
Barcelona 6
Tel: 321.2097 Telex: 52764

SWEDEN
Ajgers Elektronik AB
Box 7052
S-172-07 Sundbyberg
Tel: 08-985475, Telex: 10526

SWITZERLAND
Elbatex AG.
Albert Zwyssigstr. 28
CH 5430 Wettingen
Tel: (056) 265671

YUGOSLAVIA
Ellyptic AG.
Postfach 174, 8036 Zurich
Tel: 33.05.89, Telex: 56835

UNITED KINGDOM

St. Albans
Semicomps Ltd.,
Wellington Road,
London Colney,
St. Albans, Herts.
Tel: Bowmans Green 24522

Kelso
Semicomps Northern Ltd.
East Bowmont Street,
Kelso, Roxburghshire
Tel: Kelso 2366, Telex: 72692

Keighley
Semicomps Northern Ltd.,
Ingrow Lane,
Keighley, W. Yorks.
Tel: Keighley 65191, Telex: 517343
Portsmouth
SDS Components Ltd.,
Hilsea Industrial Estate,
Portsmouth, Hants. PO3 5JW
Tel: 0705 65311
West Drayton
Semiconductor Specialists Ltd.,
Premier House, Fairfield Road,
Yiewsley, West Drayton, Middlesex.
Tel: West Drayton 46415

MIDDLE EAST

ISRAEL
Alexander Schneider Ltd.,
44 Petach Tikva Road,
Tel-Aviv
Tel: 320.89-346.07, Telex: 033/613
Cable: DANYGAL, Tel-Aviv

IRAN
A. Ardehali S.A. Company
138 Vozara Ave., Tehran
Tel:622896

ASIA

HONG KONG
Astec Components Ltd.,
6 Hankow Road - 2nd Floor
Keystone House, Kowloon
Tel: 3-687760 Kowloon
Telex: 780-74899

INDIA
SDM and Associates
S-19 Greater Kailash-1
New Delhi-110048
Tel: New Delhi 611513

AUSTRALIA

R and D Electronics (PTY) Ltd
23 Burwood Road,
Burwood, Victoria
Tel: 288-8232, Telex: 33288

G.E.S. (PTY) Ltd.,
99 Alexander Street,
Crows Nest, N.S.W.
Tel: 439-2488, Telex: 25486

SOUTH AFRICA

Metlionics (PTY) Ltd.
P.O. Box 39690
Bramley 2018
Tel: 40-7746, Telex: 43-4852

Pace Electronic Components (PTY) Ltd
P.O. Box 6054
Dunswart 1508
Tel: 52-7545 Telex: 8-6959 SA

Data Sheet

APRIL, 1976

SERIES 8000
MICROPROCESSOR

GENERAL INSTRUMENT CORPORATION • MICROELECTRONICS

8 Bit Microprocessor System

(comprising LP 8000, LP 6000, LP 1030, LP 1010, LP 1000)

FEATURES

* 2 CHIP MINIMUM SYSTEM (plus clock)
* 48 ACCESSIBLE 8 BIT INTERNAL REGISTERS
* 48 BASIC INSTRUCTIONS
* BINARY AND DECIMAL ARITHMETIC CAPABILITY
* DIRECT AND INDIRECT INPUT OUTPUT CAPABILITY
* AUTOMATIC SUBROUTINE NESTING ON MEMORY DEVICES
* FAMILY OF DEVELOPMENT DEVICES

DESCRIPTION

The Series 8000 Logic Processor System is designed to perform any digital function using far fewer packages than a TTL or CMOS implementation. Typically a 100 package system can be reduced to a three chip solution of LP 8000 Processor, LP 6000 Program Memory and LP 1030 Clock Generator (two 40 lead DIP plus one 8 lead DIP). The consequent savings in development and production costs and increased reliability give the user many of the advantages of a customised LSI solution but without the restriction that it must be a high volume product.

The System is fabricated with General Instrument's P-channel Nitride Process which has a proven reliability and production history. All members of the Series 8000 family including Read Only Memories, General Purpose Input-Output and Memory Interface parts are fully compatible with each other.

The LP 8000 Logic Processor Unit itself is a complete 8-bit single chip MOS-LSI Microprocessor. It has a modern computer architecture with forty eight general purpose internal registers. This, coupled with a binary and decimal capability arithmetic unit, allows a versatile and sophisticated implementation of a microcomputer system. The 8-bit Data highway is supplemented by a 6-bit Address bus to give a 14-bit address capability which permits access to 16,384 words in combination of program memory, data memory or peripheral devices. The address space consists of 64 "modules" which can be either 256 words of memory or one 8-bit bidirectional I/O port.

DEVELOPMENT FAMILY

A production system may consist of only a Logic Processor, LP 8000, and ROM, LP 6000, (plus Clock Generator, LP 1030). However, the practical development family which complements the LP 8000 allows the user to implement his hardware and software in a real time replacement mode for his final mask programmed product. LP 1000 and LP 1010 parts, plus PROM, can directly replace the LP 6000 ROM. Indeed the development family may well be used as the complete solution for short run multi variety systems.

LOGIC PROCESSOR - LP (Part number LP 8000)

The logic processor (LP) is the heart of the Series 8000 system. It performs all of the arithmetic and logical functions required and also controls all activities occuring in the system. It has 48 x 8 bit working registers and an 8 bit input/output interface to which external peripherals may be attached directly.

PROGRAM MEMORY - PM (Part number LP 6000)

The program memory (PM) contains a 1K x 8 bit memory which stores the user's program. This chip also includes the program counter which points to the current address. It is arranged at the top of a four word hardware stack which is controlled by the LP for subroutine nesting. Two directly addressable 8 bit I/O interfaces are included, so that a minimum system consisting of one LP and one PM has 24 I/O leads. Extra PMs can be connected to the main system bus, up to a maximum of 16K words. Each PM and I/O interface can be addressed by the LP, the module addresses being programmed at the same time as the customer's program.

MEMORY INTERFACE CHIP - MIC (Part number LP 1000)

The memory interface (MIC) consists of an 11 bit program counter (Q register) at the top of a four word hardware stack. The address outputs are TTL compatible and enable the MIC to be interfaced directly to any external memory of up to 2048 x 8 bits. The memory can be implemented with standard parts such as PROM, RAM, core, diode matrix etc.

The memory area can be extended by using several MIC/external memory combinations. The addresses are selected by hardwiring pins to Vgg or Vcc.

INPUT/OUTPUT BUFFER - IOB (Part number LP 1010)

The input/output buffer consists of two addressable 8 bit latched I/O interfaces. The addresses are selected by hardwiring pins to Vgg or Vcc.

CLOCK GENERATOR - CG (Part number LP 1030)

The Series 8000 needs only a single phase 800KHz clock, a power-on-reset signal to clear and synchronize the system and two power supplies (+5V, -12V). Virtually all external components may be eliminated by using the clock generator (CG). The frequency of the built-in oscillator is determined by an external resistor and capacitor or can be optionally over-ridden by an input from an external oscillator. A data synchronising signal (Ø3N) is provided to act as a 'scope trigger and 'Data Valid' strobe for external hardware.

MAXIMUM RATINGS *

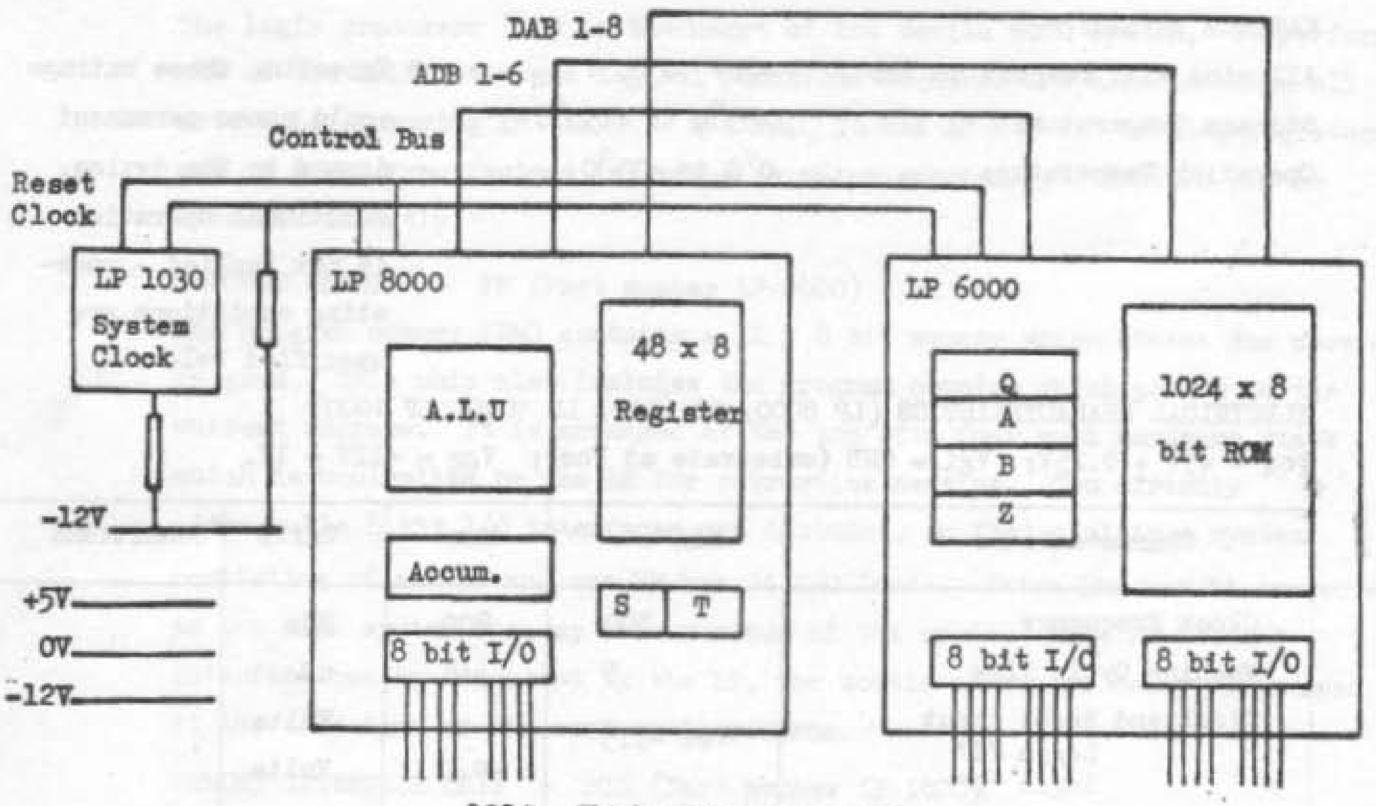
All pins with respect to Vcc	-20V to 0.3V	* Exceeding these ratings could cause permanent damage to the device. Functional operation is not implied - operating conditions are specified below.
Storage Temperature	$-55^{\circ}C$ to $+150^{\circ}C$	
Operating Temperature	$0^{\circ}C$ to $+75^{\circ}C$	

ELECTRICAL CHARACTERISTICS (LP 8000, LP 6000, LP 1010, LP 1000)

Vcc = +5V ± 0.25V; Vgi = GND (substrate at Vcc); Vgg = -12V ± 1V.

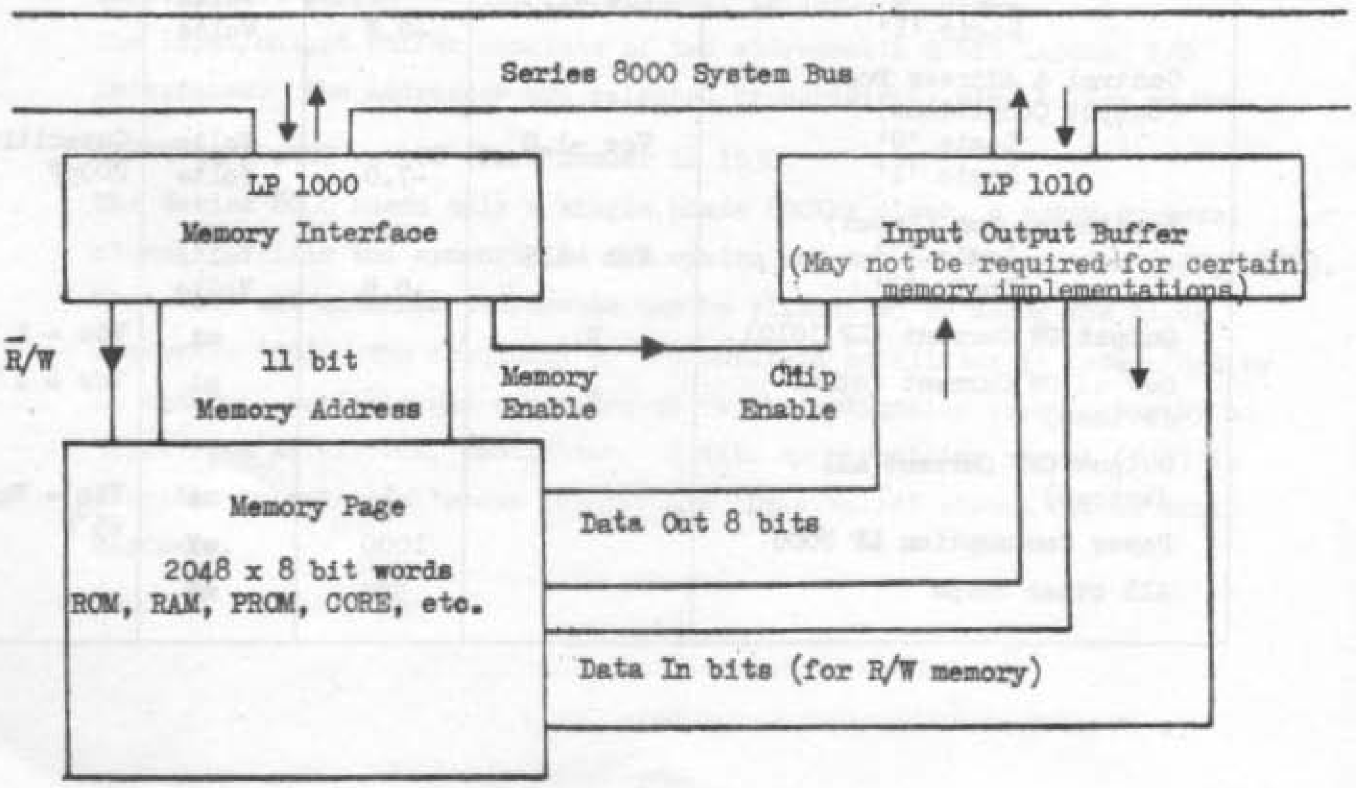
Characteristic	Min	Max	Units	Conditions
Clock Frequency	500	800	KHz	
Machine Cycle Time	5	8	uS	
Clock and Reset Input				
Logic '1'	Vcc -1.5		Volts	
Logic '0'		-9.5	Volts	
Data Bus (Input Conditions)				
Logic '1'	Vcc -1.5		Volts	
Logic '0'		+0.8	Volts	
Data Bus (Output Conditions)				
Logic '1'	Vcc -1.0		Volts	Capacitive load only, maximum 275pF
Logic '0'		+0.4	Volts	
Control & Address Bus (Input Conditions)				
Logic '0'	Vcc -1.5		Volts	
Logic '1'		+0.8	Volts	
Control & Address Bus (Output Conditions)				
Logic '0'	Vcc -1.0		Volts	Capacitive load 200pF
Logic '1'		-7.0	Volts	
Peripheral Bus (Input)				
Logic '1'	Vcc -1.5		Volts	
Logic '0'		+0.8	Volts	
Output ON Current (LP 1010)	2		mA	Vds = 1 Volt
Output ON Current (other devices)	1		mA	Vds = 1 Volt
Output OFF Current(all devices)		1	uA	Vin = Vgg at $25^{\circ}C$
Power Consumption LP 8000		1000	mW	
All other chips		500	mW	

COMPLETE SERIES 8000 MINIMUM SYSTEM



1024	Words Program Memory
48	8 bit General Registers
24	User Input Output Pins
4	Deep Address Stack

MEMORY IMPLEMENTATION FOR DEVELOPMENT SYSTEMS



Processor Signals

DAB 1-8	Bidirectional 8-lead precharged data bus, used in conjunction with address bus to implement 14-bit address word.
ADB 1-6	Push pull 6-lead address bus. This 6-bit word specifies the memory 'module' address and the 8-bit data bus specifies the 1 of 256 'intra module address'.

Processor Controls

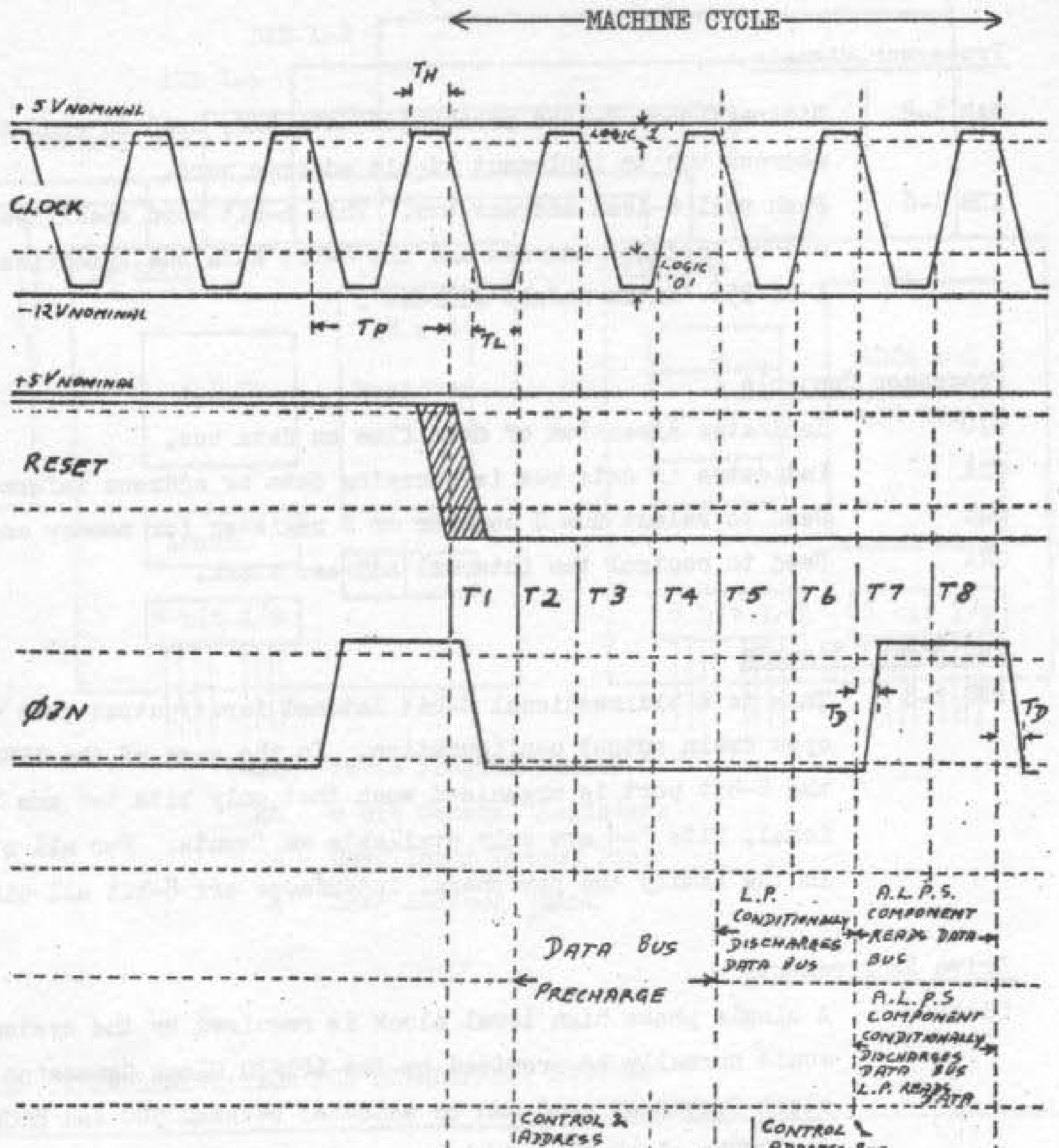
CIO	Indicates direction of data flow on data bus.
CDA	Indicates if data bus is carrying data or address information.
CQZ	Used to select the Q counter or Z register for memory addressing.
CRA	Used to control the internal address stack.

Peripheral Signals

PEB 1-8	This is a bidirectional 8-bit latched input/output port with an open drain output configuration. In the case of the LP8000 chip the 8-bit port is organised such that only bits 5-8 are bidirectional, bits 1-4 are only available as inputs. For all other chips in the family the peripheral interfaces are 8-bit all bidirectional.

Drive Requirements

Clock	A single phase high level clock is required by the system and this would normally be provided by the LP1030 Clock Generator. The clock frequency used can be selected between 500 and 800KHz. With an 800KHz clock the machine provides a 5μSec machine cycle time.
Reset	This is a clock synchronised high level signal, normally provided by the LP1030 Clock Generator.
Vcc	+5 Volt supply
Vgi	0 Volt (GND) supply
Vgg	-12 Volt supply



MACHINE CYCLE
T_H
+5V NOMINAL
LOGIC '1'
CLOCK
LOGIC '0'
-12V NOMINAL
T_P
T_L
+5V NOMINAL
RESET
T1
T2
T3
T4
T5
T6
T7
T8
Ø3N
T_D
T_D
DATA BUS
PRECHARGE
L.P. CONDITIONALLY DISCHARGES DATA BUS
A.L.P.S. COMPONENT READS DATA BUS
A.L.P.S COMPONENT CONDITIONALLY DISCHARGES DATA BUS L.P. READS DATA
CONTROL & ADDRESS BUS PRECHARGE
CONTROL ADDRESS BUS STABLE
T_D

SYSTEM TIMING AND CLOCK WAVEFORMS

CLOCK WIDTH HIGH	TH	450nSec MIN 1μsec MAX.
CLOCK WIDTH LOW	TL	450nSec MIN 1μsec MAX.
Ø 3N DELAY	TD	300 nSec MAX.
CONTROL & ADDRESS	TD	300 nSec MAX.

Mnemonics	Operation	Cycles	Comments
LAR	Load Accumulator from Register	1	These instructions are used to manipulate the contents of the accumulator with one of the 48 internal registers. They have a four bit argument and direct addressing is assumed for 0-11 but indirect for 12, 13, and 14. For indirect addressing the register address is held in S,T. Argument 12 gives register pointed to by S & T; 13 gives the same then S is decremented,14 also addresses via S & T and then S is incremented.
SAR	Store Accumulator in Register	1	
DEC	Decrement Register by one *	1	
ADR	BCD Add Accumulator with Register *	2	
BAD	Binary Add Accumulator with Register	1	
AND	Logical AND Accumulator with Register	1	
EOR	Exclusive OR Accumulator with Register	1	
	* The results of these operations are stored in the respective register. The result of all other operations is stored in the accumulator.		
LSS	Load S with Short (3-bit) Literal	1	
LST	Load T with Short (3-bit) Literal	1	
SAT	Store Accumulator in Register T	1	Lower order bits (1-3) of accumulator are copied in register T.
SST	Store Accumulator in Registers S & T	1	Bits (1-3) of accumulator copied in register S, bits (4-6) copied in register T.
LAL	Load Accumulator with 8-bit Literal	2	
LAS	Load Accumulator with 4-bit Literal	1	
ALL	Logical AND, Accumulator with 8-bit Literal	2	
ORL	Logical OR, Accumulator with 8-bit Literal	2	
EOL	Exclusive OR, Accumulator with 8-bit Literal	2	
ALA	Add Accumulator with 8-bit Literal	2	
CMP	Compare Accumulator with 8-bit Literal	2	
LIX	Load Accumulator Indirect Module X	4	The lower six bits of the X and Y registers are used to address 256-bit modules of data and program respectively. The 8-bit data bus is used to provide the intra-module address. These three instructions respectively fetch or store data using the address in the register to specify the module.
LIY	Load Accumulator Indirect Module Y	4	
SIX	Store Accumulator Indirect Module X	3	

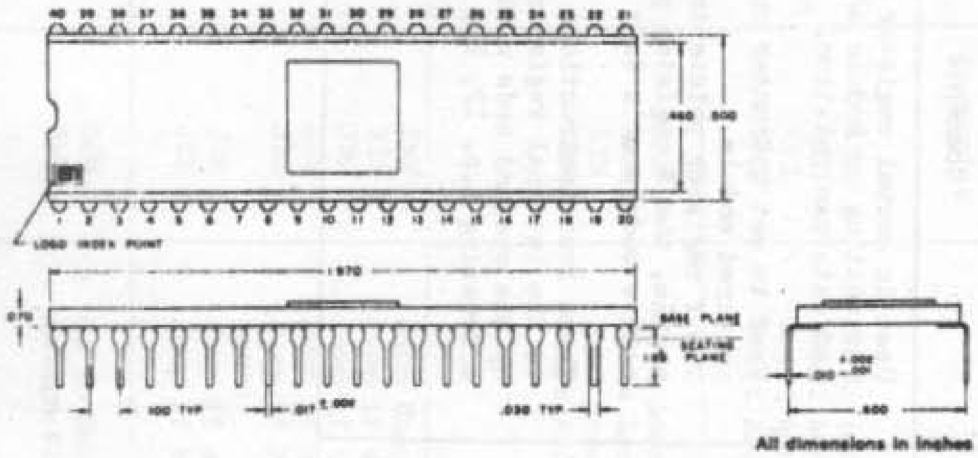
	Mnemonics	Operation	Cycle	Comments
Shift Operations	LSA	Shift Accumulator Left 1-bit	1	Carry Flag set unconditionally
	RSA	Shift Accumulator Right 1-bit	1	Carry Flag cleared uncon- ditionally
	LSN	Shift Accumulator Left 4-bits	1	
	RSN	Shift Accumulator Right 4-bits	1	
Input/Output Instructions	LAM	Load Accumulator from Module Direct	2	
	SAM	Store Accumulator in Module Direct	3	
	LIM	Load Accumulator from Module Indirect	4	The indirect I/O operations use the X register for module addressing. The SAX inst- ruction is used to set up the system for the indirect mode.
	SIM	Store Accumulator in Module Indirect	3	
Jumps within 2K Page	JMP	Jump Unconditional	3	
	JIZ	Jump if all Zeros	3/2*	
	JNZ	Jump if not all zeros	3/2	
	JIP	Jump if sign bit positive	3/2	
	JRS	Jump if Register S not equal to seven	3/2	Used to sequence through a 'page' of internal registers.
	JCS	Jump if carry bit set	3/2	
	JCN	Jump if carry bit not set	3/2	
Subroutine Instructions	GOS	Go to Subroutine	3	Program counter automatically stored in memory chip stack.
	RET	Return from Subroutine	2	

* Three cycles if true, two cycles if false.

Mnemonics	Operation	Cycle	Comments
SAX	Store Accumulator in Register X	1	Used in normal register operation and also for setting up module addresses for program and data manipulation.
SAY	Store Accumulator in Register Y	1	
LAX	Load Accumulator from Register X	1	
LAY	Load Accumulator from Register Y	1	
SZX	Store Accumulator in Z Register Module X	3	Used to set Q Counter or Z register of required module. The Y register points to the active program module, the X register points to an alternative module or a data space.
SZY	Store Accumulator in Z Register Module Y	3	
SQZ	Store Accumulator in Q Counter Module X	3	
SQY	Store Accumulator in Q Counter Module Y	3	
SAV	Store Accumulator in Register V	1	These four instructions allow direct access to the internal registers which are masked by the operand code used for indirect addressing i.e. 12, 13,
SAW	Store Accumulator in Register W	1	
LAV	Load Accumulator in Register V	1	
LAW	Load Accumulator in Register W	1	
CIA	Clear Accumulator to Zeros	1	

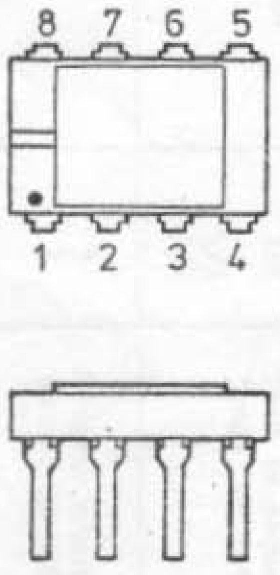
PACKAGE: LP 8000, LP 6000, LP 1010, LP 1000

40 Lead Dual In Line.



PACKAGE: LP 1030 Clock and Power On Reset Generator.

8 Lead Dual In Line.



Pin Connections - LP 1030

1	V_{cc}
2	Timing Input
3	Synchronising Input
4	Reset Input
5	V_{gg}
6	Clock Output
7	ϕ_{3N} Output
8	Reset Output

LP8000 LOGIC PROCESSOR

Pin Connections

1	Vcc	21	Peripheral Bus 1
2	Power on Reset	22	Address Bus 1
3	Clock	23	" " 2
4	Not Used	24	" " 3
5	" "	25	" " 4
6	Data Bus 8	26	" " 5
7	" " 7	27	" " 6
8	" " 6	28	Not Used
9	" " 5	29	" "
10	" " 4	30	" "
11	" " 3	31	" "
12	" " 2	32	" "
13	" " 1	33	" "
14	Peripheral Bus 8	34	" "
15	" " 7	35	CIO
16	" " 6	36	CDA
17	" " 5	37	CQZ
18	" " 4	38	CRA
19	" " 3	39	Vgg
20	" " 2	40	Vgi

LP6000 PROGRAM MEMORY

Pin Connections

1	Vcc	21	CQZ
2	Data Bus 1	22	CRA
3	" " 2	23	Vgg
4	" " 3	24	Vgi
5	" " 4	25	Peripheral Bus B8
6	" " 5	26	" " B7
7	" " 6	27	" " B6
8	" " 7	28	" " B5
9	" " 8	29	" " B4
10	Not Used	30	" " B3
11	Power on Reset	31	" " B2
12	Address Bus 1	32	" " B1
13	" " 2	33	Peripheral Bus A8
14	" " 3	34	" " A7
15	" " 4	35	" " A6
16	" " 5	36	" " A5
17	" " 6	37	" " A4
18	Clock	38	" " A3
19	CDA	39	" " A2
20	CIO	40	" " A1

LP1000 MEMORY INTERFACE

Pin Connections

1	Vcc	21	CDA
2	Data Bus 8	22	CQZ
3	" " 7	23	CRA
4	" " 6	24	MIC Enable
5	" " 5	25	Memory Enable
6	" " 4	26	$\overline{R}$/W
7	" " 3	27	Clock
8	" " 2	28	Address Bit 1
9	" " 1	29	" " 2
10	Power on Reset	30	" " 3
11	Address Bus 1	31	" " 4
12	" " 2	32	" " 5
13	" " 3	33	" " 6
14	" " 4	34	" " 7
15	" " 5	35	" " 8
16	" " 6	36	" " 9
17	PAD 4	37	" " 10
18	PAD 5	38	" " 11
19	PAD 6	39	Vgg
20	CIO	40	Vgi

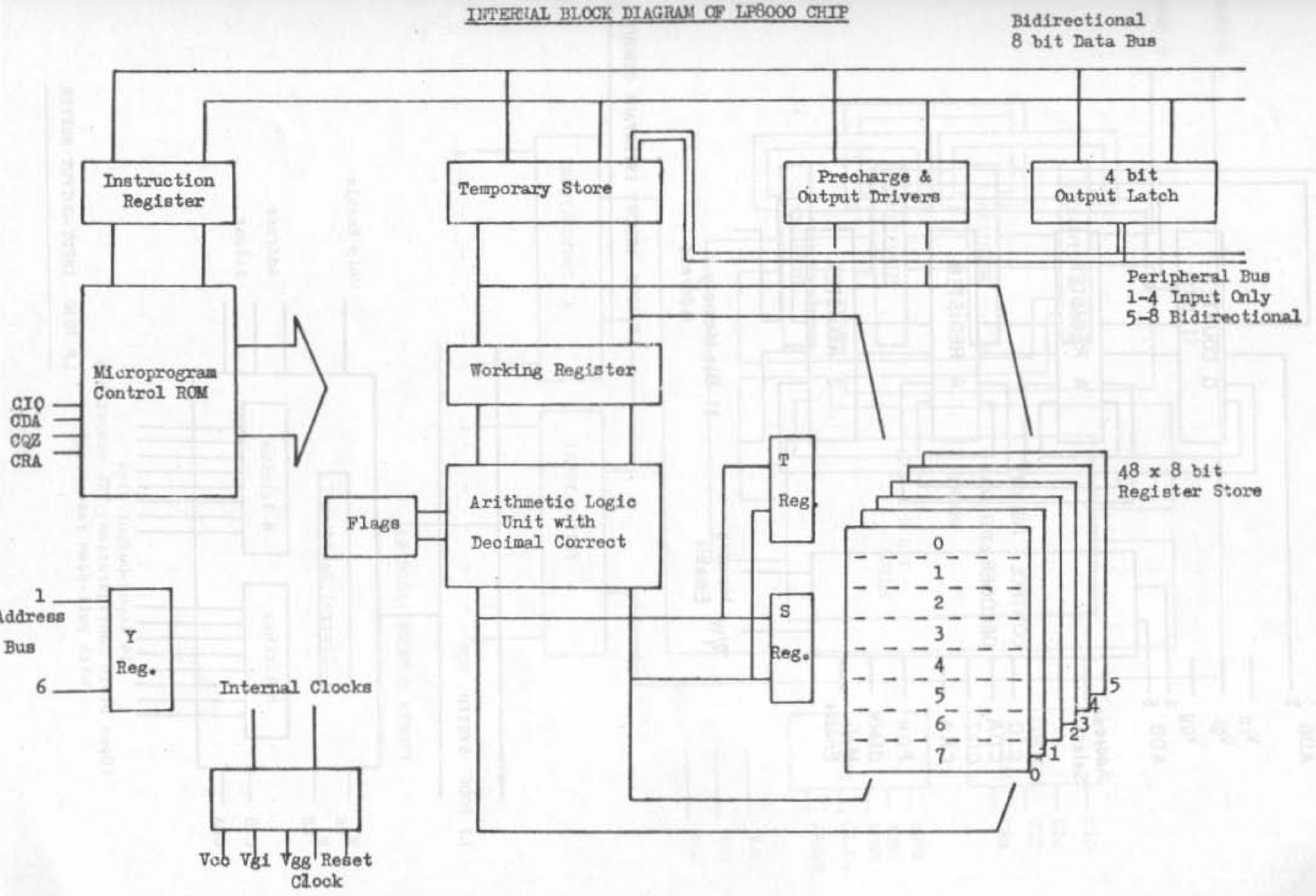
LP1010 INPUT/OUTPUT BUFFER

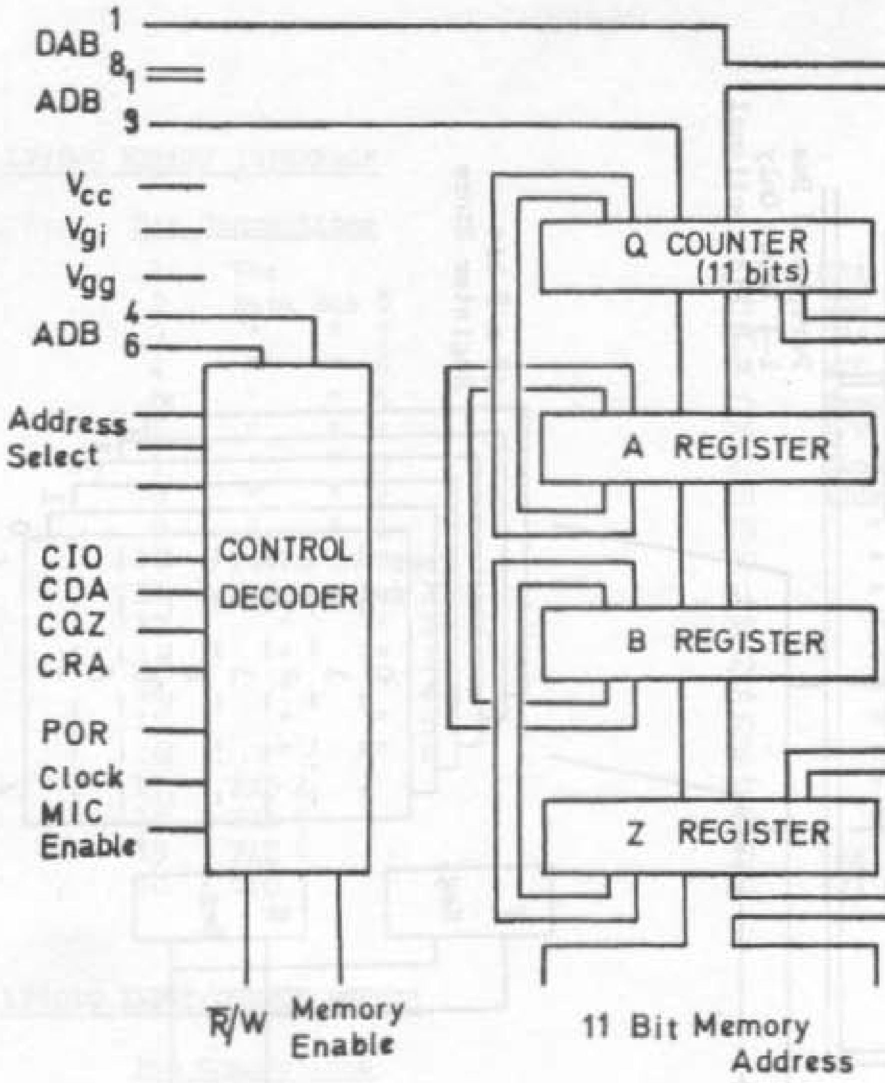
Pin Connections

1	Vcc	21	PAD 2
2	Data Bus 1	22	PAD 3
3	" " 2	23	Vgg
4	" " 3	24	PAD 4
5	" " 4	25	Peripheral Bus B8
6	" " 5	26	" " B7
7	" " 6	27	" " B6
8	" " 7	28	" " B5
9	" " 8	29	" " B4
10	Chip Select	30	" " B3
11	Power on Reset	31	" " B2
12	Address Bus 1	32	" " B1
13	" " 2	33	Peripheral Bus A8
14	" " 3	34	" " A7
15	" " 4	35	" " A6
16	" " 5	36	" " A5
17	" " 6	37	" " A4
18	Clock	38	" " A3
19	CDA	39	" " A2
20	CIO	40	" " A1

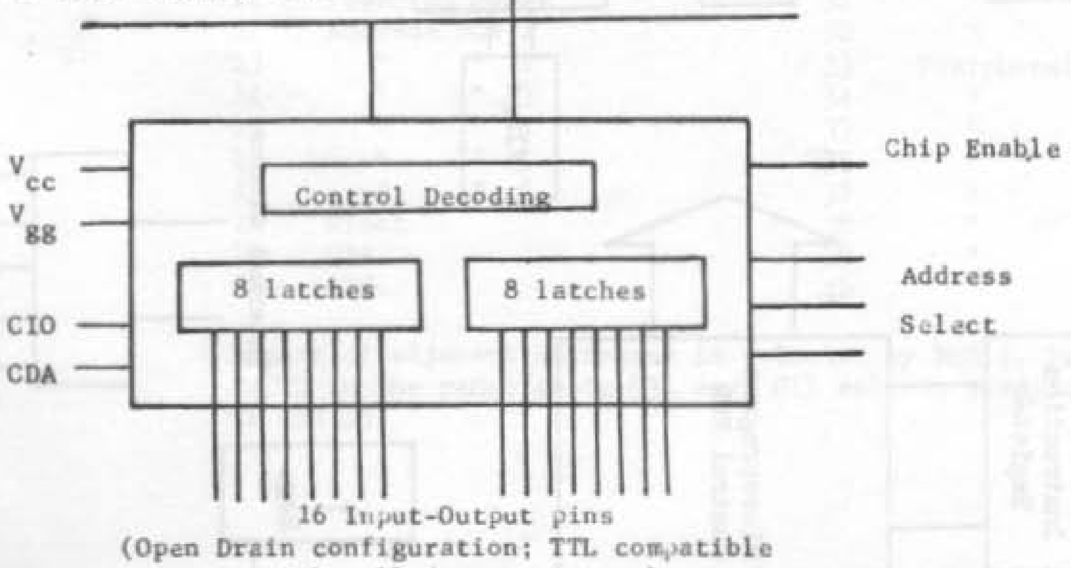
A pair of adjacent addresses is selected by PAD 4, PAD 3 and PAD 2 in the range 48 to 63, e.g. 011 selects peripheral addresses 54 and 55.

INTERNAL BLOCK DIAGRAM OF LP8000 CHIP

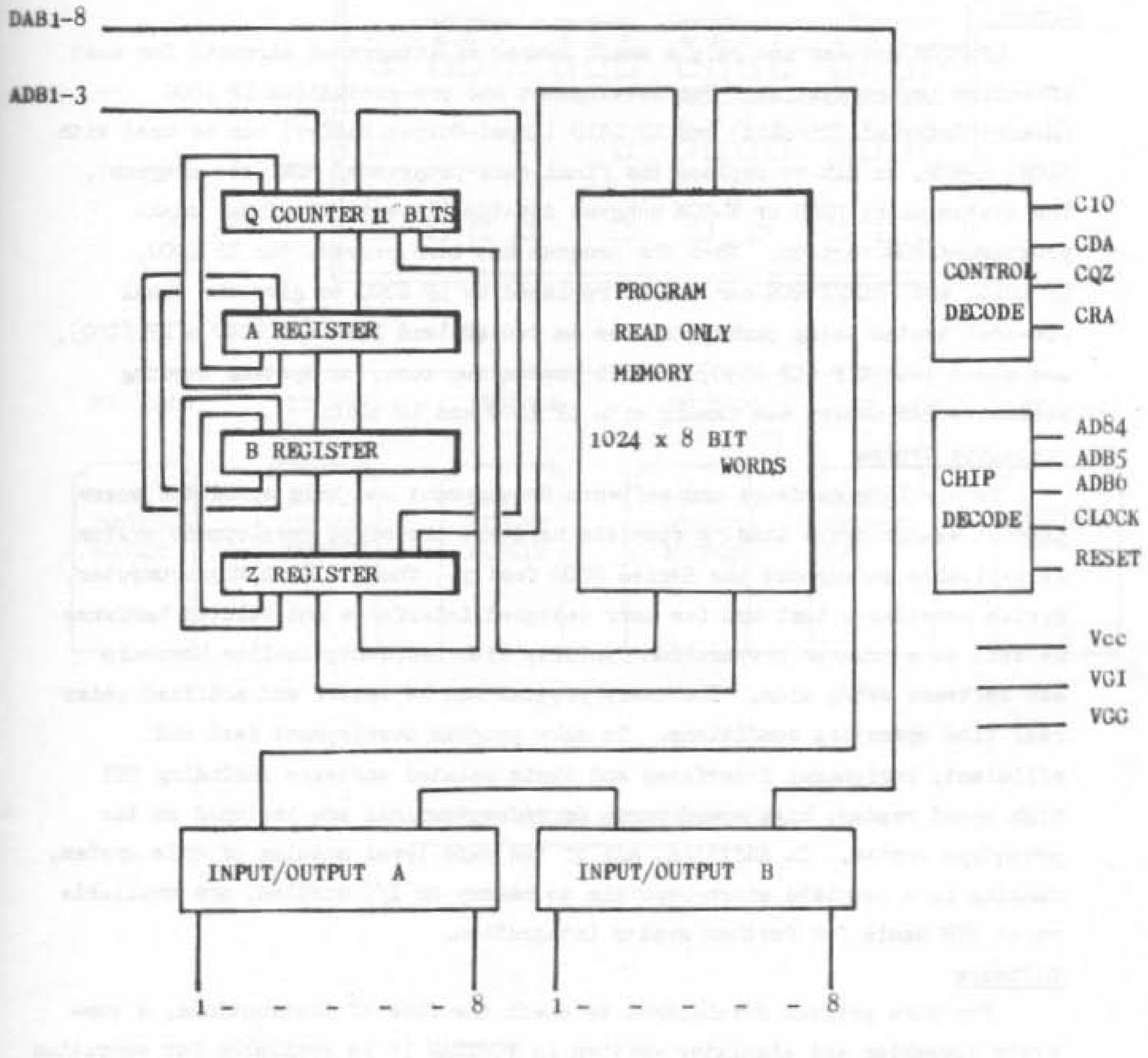




LP 1000 MEMORY INTERFACE CIRCUIT



LP 1010 INPUT-OUTPUT BUFFER



LP 6000 PROGRAM MEMORY

DEVELOPMENT SUPPORT

Circuits

LP 8000 systems use only a small number of integrated circuits for cost effective implementation. For development and pre-production LP 1000 (Memory Interface Circuit) and LP 1010 (Input-Output Buffer) can be used with PROM, E-ROM, or RAM to replace the final mask-programmed ROM (see diagram). The system using PROM or E-ROM behaves identically with the final mask-programmed ROM version. When the program has been proved, the LP 1000, LP 1010, and PROM/E-ROM can all be replaced by LP 6000 to give the final low-cost system using perhaps as few as two 40 lead DIPs (LP 8000 + LP 6000), and one 8 lead DIP (LP 1030). Small production runs, or systems needing extensive RAM memory can remain with LP 1000 and LP 1010.

Prototype System

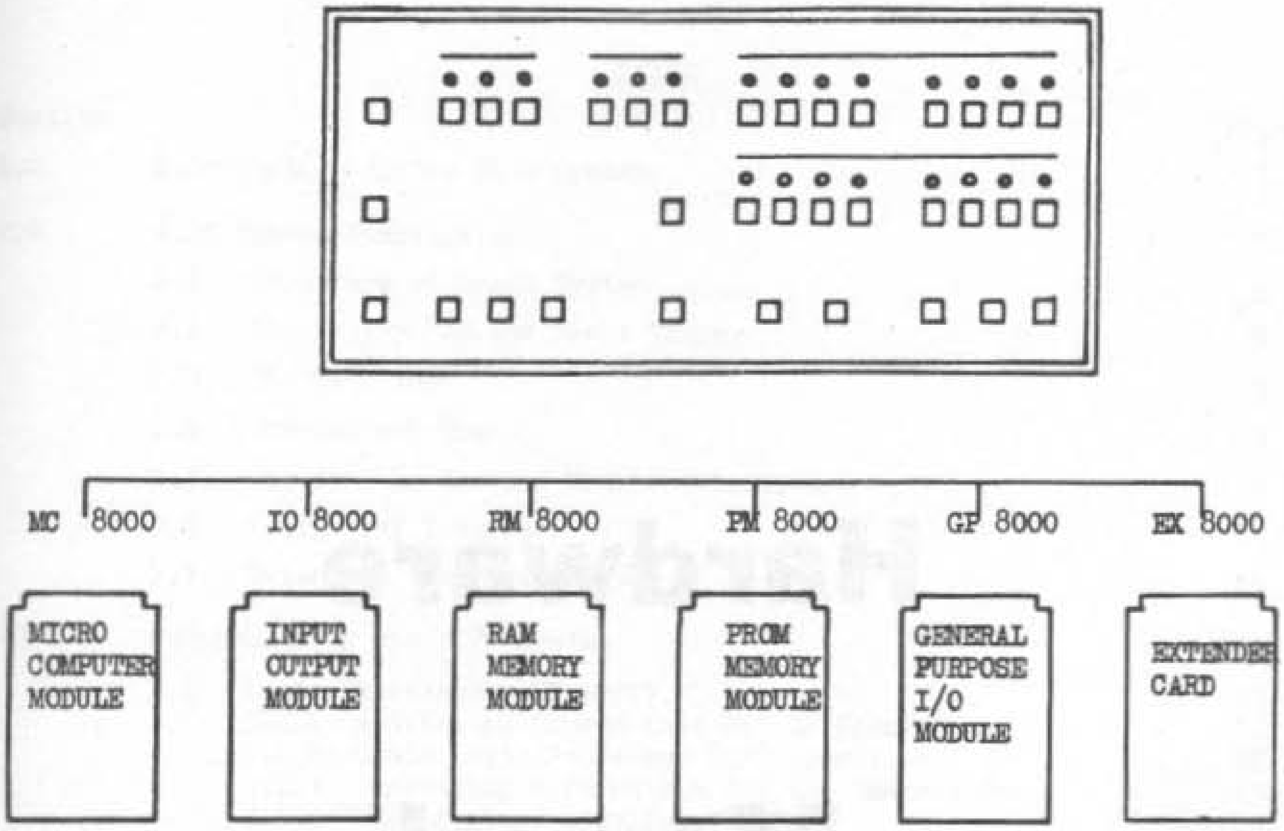
To simplify hardware and software development and help speed the users product design cycle time, a complete hardware prototype development system is available to support the Series 8000 family. The GIC 8000 Microcomputer System provides a test bed for user designed interfaces and related hardware as well as a program preparation facility with resident, on-line hardware and software debug aids. The users program can be tested and modified under real time operating conditions. To make program development fast and efficient, peripheral interfaces and their related software including TTY high speed reader, high speed punch or video terminal are included on the prototype system. In addition, all of the card level modules of this system, ranging from complete microcomputers to memory or I/O modules, are available on an OEM basis for further system integration.

Software

For pure program development to check the flow of instructions, a complete assembler and simulator written in FORTRAN IV is available for operation on minicomputer systems or on internal or external time share networks.

Manual

A manual describing complete hardware aspects of Series 8000, and details of the program preparation software is available from all General Instrument Microelectronics Sales Offices, Agencies, and Distributors.



SERIES 8000 MICROCOMPUTER

Hardware

Manual

INDEX

LIST OF ILLUSTRATIONS

THE ALPS SYSTEM HARDWARE MANUAL

1.0 Introduction to the ALPS System

General Instrument Microelectronics Advanced Logic Processing System (ALPS) is an 8-bit parallel processing system designed for general purpose logic and calculation problems. Its instruction set has been chosen so that common sub-routines may be carried out with the minimum length of program. The hardware has also been optimised so that the user needs the minimum number of external components. A simple system can be built with only two chips and a clock generator. Dedicated input/output interfaces are provided so that there is no need for complicated external bus multiplexing.

All the components of the ALPS set have been designed to be totally compatible with each other. They are manufactured using the GIANT nitride process, known for its high stability and reliability.

The set consists of

(i) LOGIC PROCESSOR - LP (Part number LP 8000)

The logic processor (LP) is the heart of the ALPS system. It performs all of the arithmetic and logical functions required and also controls all activities occuring in the ALPS system. It has 48 x 8 bit working registers and an 8 bit input/output interface to which external peripherals may be attached directly.

(ii) PROGRAM MEMORY - PM (Part number LP 6000)

The program memory PM contains a 1K x 8 bit memory which stores the user's program. This chip also includes the program counter which points to the current address. It is arranged at the top of a four word hardware stack which is controlled by the LP for subroutine nesting. Two directly addressable 8 bit I/O interfaces are included, so that a minimum system consisting of one LP and one PM has 24 I/O leads. Extra PMs can be connected to the main system bus, up to a maximum of 16K words. Each PM and I/O interface can be addressed by the LP, the module addresses being programmed at the same time as the customer's program.

(iii) MEMORY INTERFACE CHIP - MIC (Part Number LP1000)

The memory interface chip consists of an 11 bit program counter at the top of a four word hardware stack. The address outputs are TTL compatible and enable the MIC to be interfaced directly to any external memory of up to 2K x 8 bits. The memory plane may be implemented with standard chips such as PROMS and RAMS or cores and diode matrixes can also be used. A memory area implementation with the MIC chip will have an analogous operation to the memory contained in the LP6000 mask programmed memories but including the facility of read/write operation.

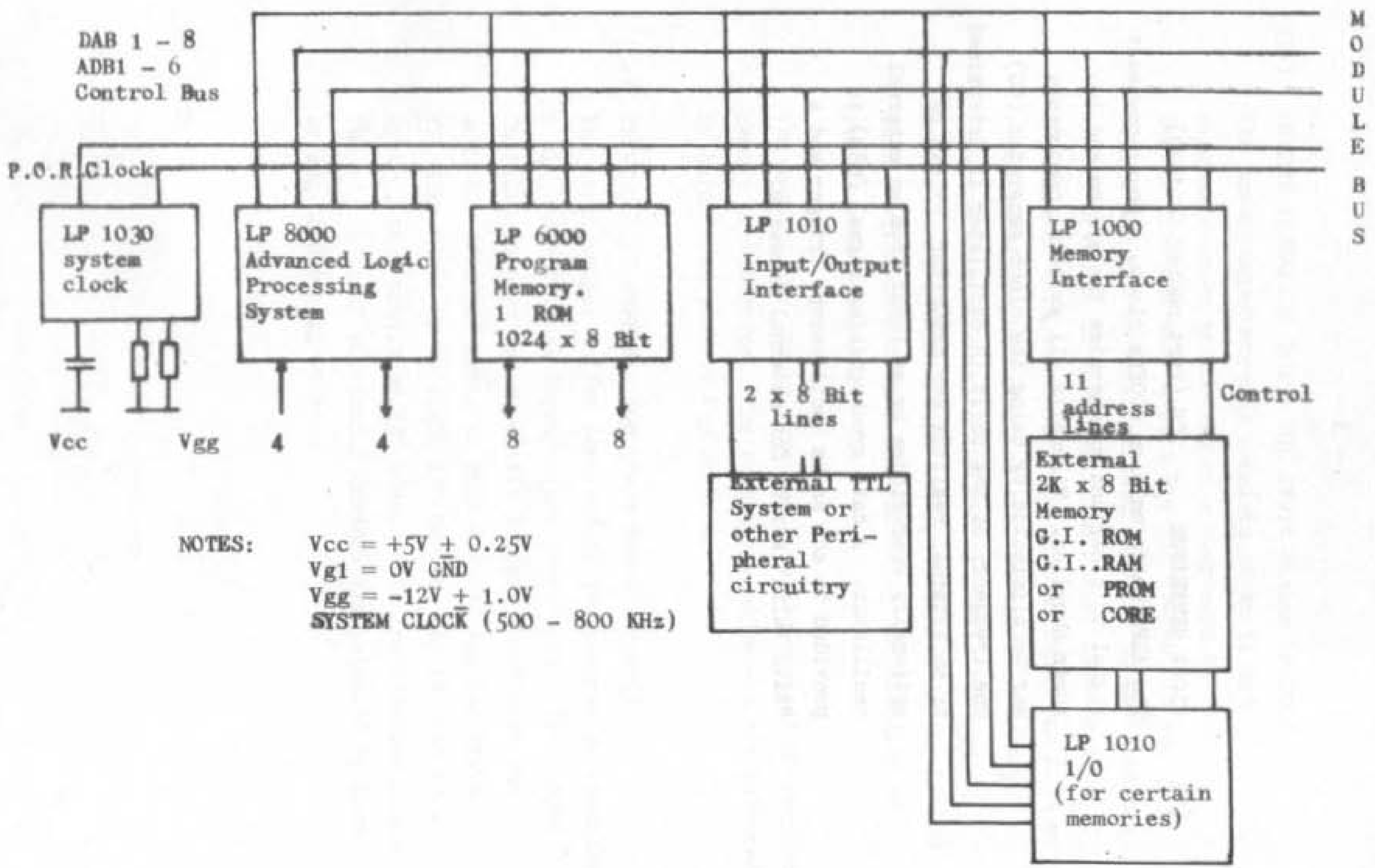
The memory area can be extended by using several MIC/external memory combinations. The particular addresses are selected by hardwiring pins to Vgg or Vcc.

(iv) INPUT/OUTPUT BUFFER - IOB (Part Number LP1010)

The input/output buffer consists of two program addressable 8 bit bidirectional input/output interfaces. The output facility is provided by static latches which with the addition of a 6K8 resistor pull down to Vgg can drive TTL directly. Up to eight I/O devices may be used in a system, thus providing 128 addressable input/output lines. The addresses of particular devices are selected by hard-wiring pins to Vgg or Vcc.

(v) CLOCK GENERATOR - CG (Part number LP 1030)

The ALPS needs only an 800KHz clock, a power-on-reset signal to clear and synchronize the system and two power supplies. Virtually all external components may be eliminated by using the clock generator (CG). The frequency of the built in oscillator is determined by an external resistor and capacitor, or can be optionally over-ridden by an input from an external oscillator. A data synchronising signal (Ø3N) is provided to act as an oscilloscope trigger and a "Data Valid" strobe for external hardware.


DAB 1 - 8
ADB1 - 6
Control Bus
P.O.R.
Clock
M O D U L E B U S
LP 1030 system clock
Vcc
Vgg
LP 8000 Advanced Logic Processing System
4
4
LP 6000 Program Memory. 1 ROM 1024 x 8 Bit
8
8
LP 1010 Input/Output Interface
2 x 8 Bit lines
External TTL System or other Peripheral circuitry
LP 1000 Memory Interface
11 address lines
Control
External 2K x 8 Bit Memory G.I. ROM G.I..RAM or PROM or CORE
LP 1010 1/0 (for certain memories)

NOTES: Vcc = +5V $\pm$ 0.25V
Vg1 = 0V GND
Vgg = -12V $\pm$ 1.0V
SYSTEM CLOCK (500 - 800 KHz)

FIG.1 BLOCK DIAGRAM OF ADVANCED LOGIC PROCESSING SYSTEM A.L.P.S.

2.0 ALPS System Description

2.1 STRUCTURE OF BASIC SYSTEM

ALPS is a modular system which is easily expanded according to the complexity required. Programs can be written so that common subroutines are contained in a basic chip set which can be used in a range of machines.

The system is built around a module bus with four basic parts. See Fig.1.

(a) Supply Bus (5 leads)

The ALPS system needs two supplies of +5V and -12V (or -5V and -17V) and an 800KHz clock. A power-on-reset signal at switch on is necessary to clear the I/O interfaces and synchronize the on-chip clock dividers on each chip.

(b) Data Bus (8 bits)

Information (addresses or data) pass to and from the LP using this bus.

(c) Control Bus (4 bits)

This bus is driven from the LP and controls the activity of all chips in the ALPS system.

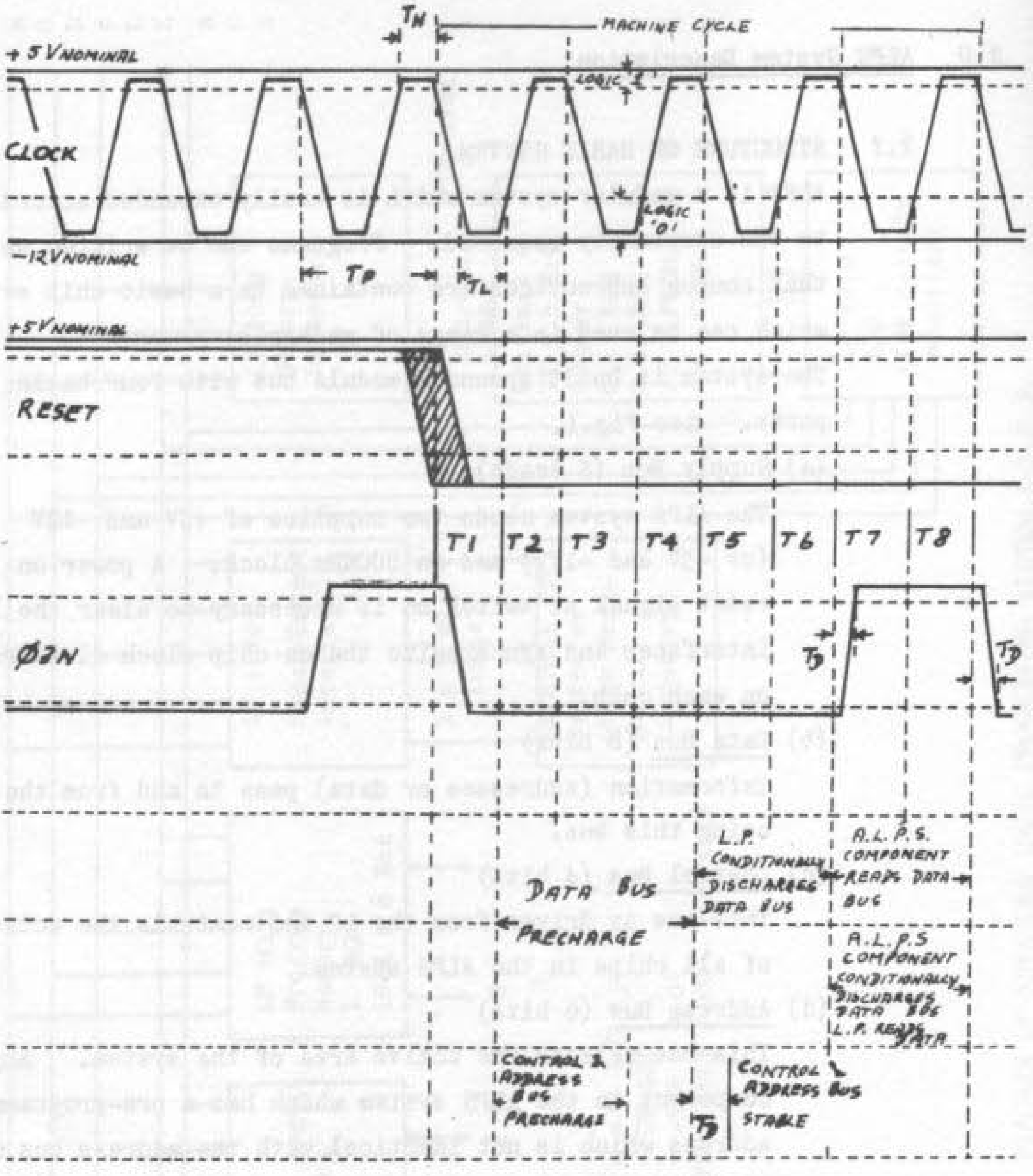
(d) Address Bus (6 bits)

This bus selects the active area of the system. Any component in the ALPS system which has a pre-programmed address which is not identical with the address bus will be inactive.

2.2 THE SUPPLY BUS AND BASIC SYSTEM TIMING

The ALPS system timing is controlled from an 800KHz clock. This is divided by four on each chip to provide a basic machine cycle of 5uS. Fig.2 shows the system clock and how the eight time slots T1-T8 are defined. The start of the negative going edges are defined as F1-F4 and are used as references to define system timing in the electrical specification (see section 7).

The power on reset signal must be synchronised with the clock as shown.



TN
MACHINE CYCLE
+5V NOMINAL
LOGIC '1'
CLOCK
LOGIC '0'
-12V NOMINAL
TP
TL
+5V NOMINAL
RESET
T1
T2
T3
T4
T5
T6
T7
T8
Ø3N
TD
TD
L.P. CONDITIONALLY DISCHARGES DATA BUS
A.L.P.S. COMPONENT READS DATA BUS
DATA BUS
PRECHARGE
A.L.P.S COMPONENT CONDITIONALLY DISCHARGES DATA BUS
L.P. READS DATA
CONTROL & ADDRESS BUS PRECHARGE
CONTROL & ADDRESS BUS STABLE
TD

FIG.2. SYSTEM TIMING AND CLOCK WAVEFORMS

CLOCK WIDTH HIGH	TH	450n Sec MIN 1μ Sec MAX.
CLOCK WIDTH LOW	TL	450n Sec MIN 1μ Sec MAX.
Ø3N DELAY	TD	300n Sec MAX.
CONTROL & ADDRESS ~~BUS DELAY~~	TD	300n Sec MAX.

2.3 THE DATA BUS

The 8 bit data bus activity is controlled by the LP using the 4 control busses (see 2.4). Information, which may be addresses or data, can be transferred: (see Fig.3)

(a) from LP to another ALPS component. In this mode the LP precharges all data busses in T2, T3, T4 and then conditionally discharges it in T5 and T6. The data is now ready to be read into the addressed ALPS component on T7 and T8.

(b) from an ALPS component to LP. In this mode the LP precharges all data busses in T2, T3 and T4. The addressed ALPS component discharges the data bus in T7, T8 and at the same time is read into the LP for processing and/or decoding.

Note that the LP always precharges the data bus every machine cycle. Many instructions do not use the data bus, e.g. 'increment accumulator' in which case the charge on the data bus remains until the next machine cycle.

The data bus normally carries 8 bits of data to and from the LP with data bus 8 as the most significant digit. Note that the data is inverted. This is only of importance when debugging a system by monitoring data bus activity.

The data bus is also used in conjunction with the 6 bit address bus to provide the least significant 8 bits of a 14 bit address. Whenever any components are connected to ALPS system it must be borne in mind that the data bus is a high impedance bus so that it is not possible to connect items which require significant input current (e.g. TTL). This is of no concern to the user who only uses ALPS system components as they have all been designed to be compatible with the data bus requirements. Components requiring input current must be connected via any of the ALPS peripheral interfaces.

2.4 THE CONTROL BUS

The control bus consists of four control signals.

CIO:	controls whether information is written onto the data bus or read from it.
CDA:	determines whether the information on the data bus is data or an address

	CIO input/output	CDA data/address	CQZ Q or Z register	CRA stack operation	TYPE OF MODULE ADDRESSED		
					ROM as in PM,or MIC with ROM	MIC with RAM	Peripheral Interface as in LP, PM,
*	0	0	0	0	instruction fetch	read data addressed by Q	data read from interface to LP
	0	0	0	1			
*	0	0	1	0	read data addressed by Z	read data addressed by Z	
	0	0	1	1			
*	0	1	0	0	No operation	No operation	no operation
*	0	1	0	1	Return from sub-routine	Return from sub-routine	
	0	1	1	0	no operation	No operation	
	0	1	1	1			
*	1	0	0	0	no operation	write data into location addressed by Z	data written into interface from LP
	1	0	0	1			
	1	0	1	0			
	1	0	1	1			
*	1	1	0	0	address → Q	address -> Q	no operation
*	1	1	0	1	address → Q and Push Stack	address -> Q and Push Stack	
*	1	1	1	0	address → Z	address -> Z	
	1	1	1	1			

* Only these codes are used by LP

FIG.4 SHOWING EFFECT OF CONTROL BUS ON ALPS COMPONENTS

CQZ: controls whether the Q or Z register is uded in the PM, PMD or MIC.

CRA: controls whether the 4 word hardware stack on the PM, PMD or MIC is pushed (or popped). It differentiates between jump and subroutine instructions.

Fig.4 shows all 16 possible codes and the effect they have on the ALPS components. In practice the LP only uses the codes 0000, 0010, 0100, 0101, 1000, 1100, 1101 and 1110.

2.5 THE ADDRESS BUS AND MODULE SELECTION

All components in the ALPS system are connected to a common module bus, to which the outputs of each component are effectively wire OR'ed. The system is divided into modules for purposes of addressing and the address bus (6 bits) selects which of these 64 possible modules is active.

A module is defined as

256 x 8 bits of ROM
OR 256 x 8 bits of RAM
OR 1 input/output interface.

For calculating the maximum size system for a given application it is useful to describe the ALPS system components in terms of modules (maximum permissible 64).

Component	Module		No. of Modules
LP	1 8 bit I/O interface		1
		TOTAL	1
PM	1K x 8 bits ROM		4
	2 8 bit I/O interfaces		2
		TOTAL	6
MIC	2K x 8 bits External Memory		8
		TOTAL	8
I.O.B.	2 x 8 bit I/O interface.		2
		TOTAL	2

ALPS Component	Address Bus (defines 1 of 64 possible modules)							Data Bus							
	6	5	4	3	2	1		8	7	6	5	4	3	2	1
PM	selects 2 chips containing 4 modules each			selects chip 0 or 1	PM: decoded 1 out of 1024 to address ROM array										
MIC + 2K x 8	selects 1 MIC out of 8 possible			decoded 1 out of 2048 to address memory array											
I/O channel (on PM, or IOB).	seledts 1 out of 64 possible peripheral interfaces.							Not used for addressing: only used for data.							

FIG.5 ALPS ADDRESSING SCHEME

Intramodule selection is performed by 8 additional address bits carried on the data bus. This permits selection of 1 of 256 data word locations in a memory module. If an I/O module was accessed the data bus only carries the 8 bit data word, because there is no explicit intramodule address for an I/O location.

Memory modules are further grouped in 2K x 8 bit pages and since the PM is a basic 1K x 8 device the PM hardware has been arranged such that the chips can be used in pairs to appear as one continuous 2K x 8 bit memory. The active chip is automatically selected by the 11th bit of the system address.

The pair of chips can be considered as one 2K x 8 bit chip for programming purposes. Fig.5 illustrates the addressing scheme.

When address information has to be written into the Q or Z registers of memory chips the active PM or MIC is selected by address bus ADB4, 5 & 6. This specifies a group of 8 modules and for the PM, the address information will be written into a pair of chips, i.e. 2 x 4 modules, When reading information from a pair of PM's, ADB3 is examined to determine which of the pair the data is to come from. With a MIC chip implementation, normal address decoding selects the required memory location specified by the 11 bit address.

For the case of peripheral interfaces, these are also assigned module addresses which are specified by the 6 bit address bus. Using PM chips, any peripheral address can be selected except 0 and 63, (Module 0 must be reserved for the start of program and 63 is already allocated to the LP I/O interface. Due to pin limitations on the I.O.B. chip only three moudle address leads are provided and these are used to select pairs of I/O peripherals in the range 48-63. The three select leads are PAD2, 3, 4 and they are compared with the system address bus ADB2,3,4. The address bus ADB5,6 must be at a logic "1" to select an I/O device, this gives the range (48 + 0 -> 48 + 15). The A-B selection from the pair of peripheral parts is performed by address bus ADB1.

The normal method of allocating module addresses is to start the program ROM or RAM arrays at module 0 and work upwards, and to assign peripheral interfaces from 63 and work downwards. This method makes program writing simpler and enables easy expansion of the system.

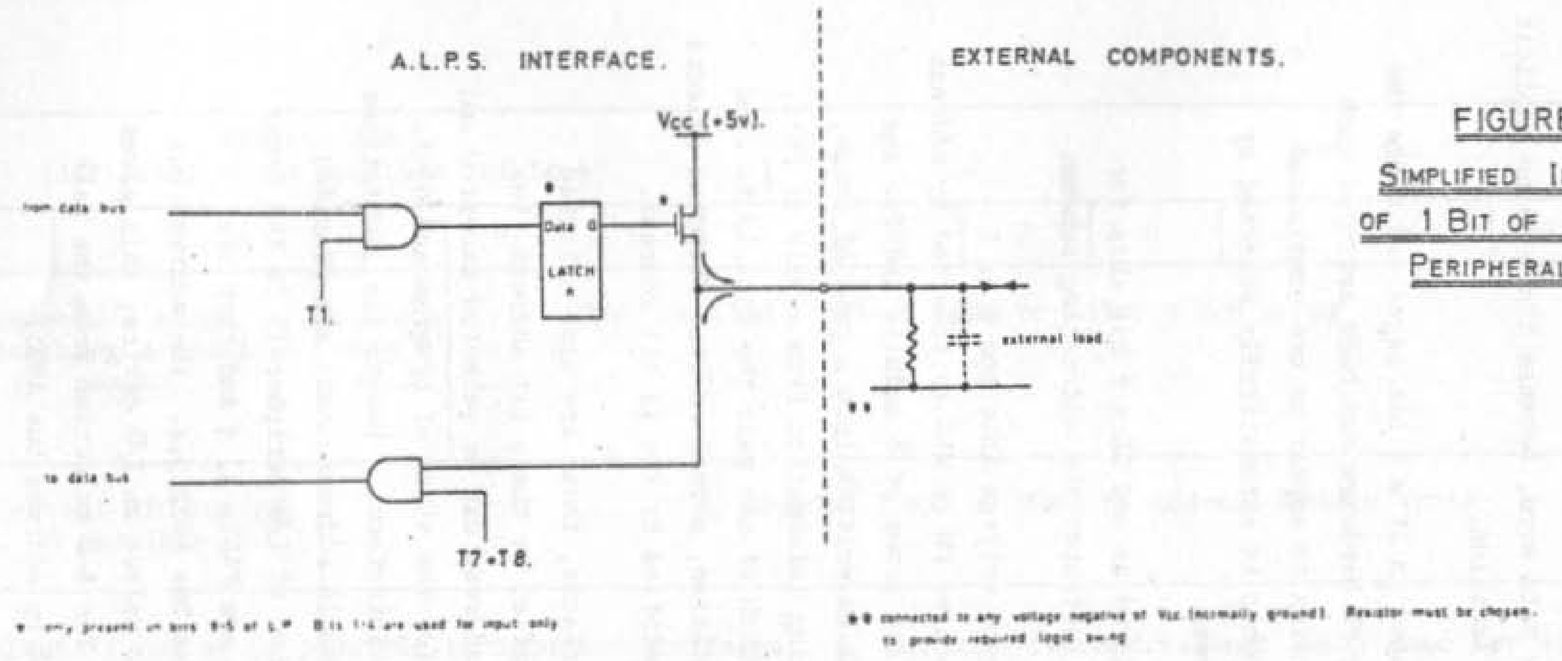

A.L.P.S. INTERFACE.
EXTERNAL COMPONENTS.
Vcc (+5v).
from data bus
to data bus
T1.
T7+T8.
Data
LATCH
external load.
* only present on bits 9-5 of L.P. Bits 1-4 are used for input only
** connected to any voltage negative of Vcc (normally ground). Resistor must be chosen to provide required logic swing

FIGURE 6.

SIMPLIFIED INTERFACE LOGIC OF 1 BIT OF A.L.P.S. PERIPHERAL INTERFACE.

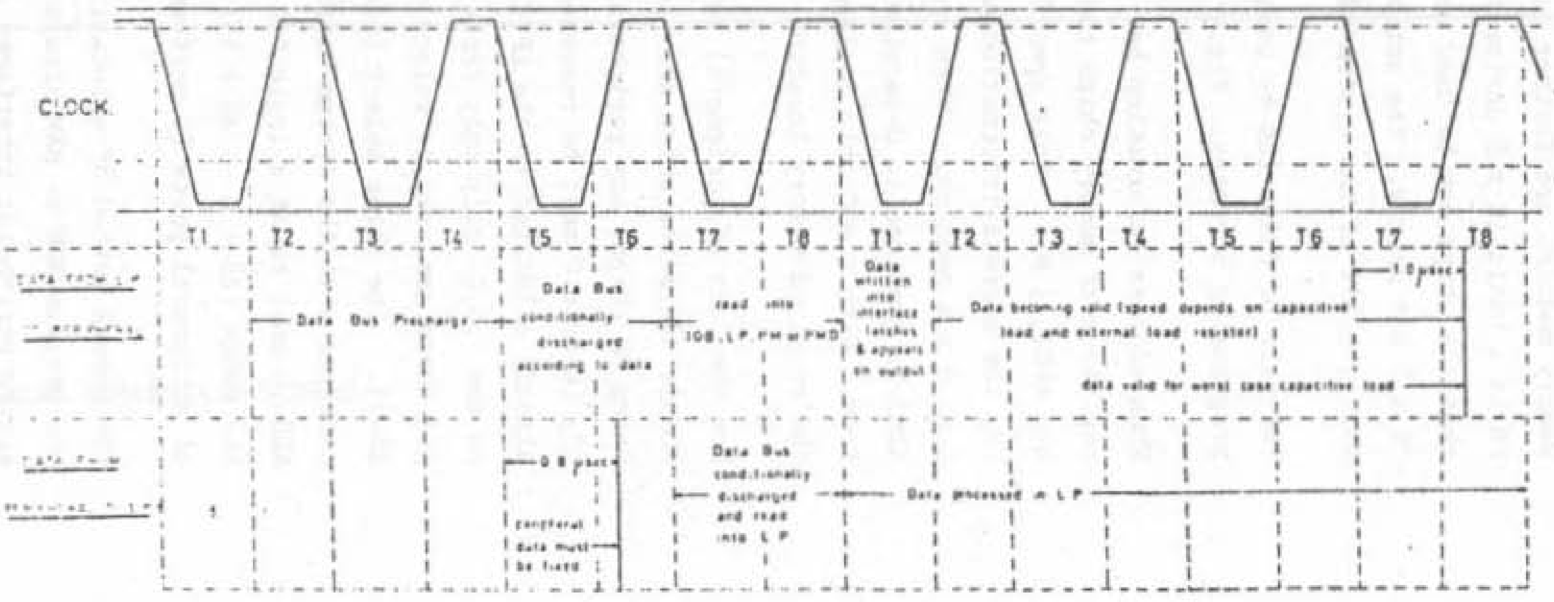

CLOCK.
T1 T2 T3 T4 T5 T6 T7 T8 T1 T2 T3 T4 T5 T6 T7 T8
Data Bus Precharge
Data Bus conditionally discharged according to data
read into IOB, L.P. PM or PMD
Data written into interface latches & appears on output
Data becoming valid (speed depends on capacitive load and external load resistor)
1.0 μsec
data valid for worst case capacitive load
0.8 μsec
peripheral data must be fixed
Data Bus conditionally discharged and read into L.P.
Data processed in L.P.

FIGURE 7.

TIMING FOR PERIPHERAL INTERFACES

2.6 PERIPHERAL INTERFACES

The input/output interfaces are identical on all ALPS components except that on the LP (module address 63, see below).

The standard 8 bit interface consists of 8 latches connected to 8 open drain MOS output transistors. The configuration is shown in Fi.g.6 & 7. When reading from the interface the pins are treated as inputs and the data is fed to the LP via the data bus. Input and outputs are effectively wire OR'ed together so that when information is to be read from the interface, it is necessary to write 0000 0000 into the latches to prevent incorrect information being read in from the latches. This is done automatically at power on. This feature can also be used to advantage. For example it is possible to use an unused interface to store an 8 bit word. I/O number 63 on the LP differs in that it can read 8 bits of data but only send four bits of data out (bits 8-5) as there are only four latches.

The peripheral interfaces can be addressed in two ways:

(i) Directly (using LAM and SAM instructions). As there is only a 3 bit data field available to specify the peripheral being addressed, 56 is added to form addresses in the range 56-63. This is convenient in practice as it is normal to allocate peripherals from module 63 working downwards while the program starts at module 0 and works upwards.

(ii) Indirectly (using LIX and SIX). Any of the I/O's may be addressed in this mode as a full 6 bit address is available stored in the X location of the scratchpad store.

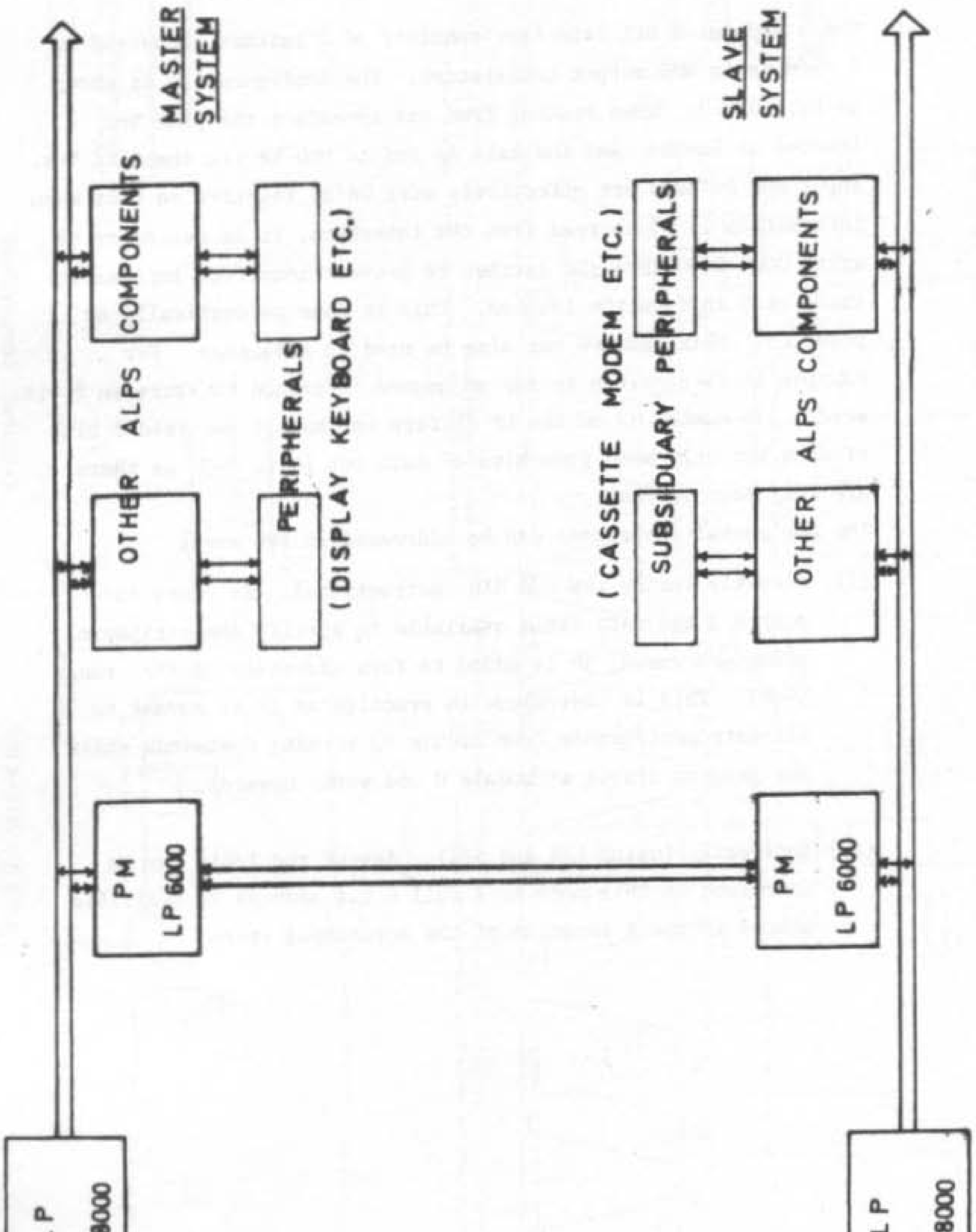


Figure 8 ADDING ANOTHER LOGIC PROCESSOR

3. COMPONENTS OF THE ALPS SYSTEM

3.1 LOGIC PROCESSOR (LP)

3.1.1 Addressing Scheme used for LP Scratchpad.

The LP contains 48 8 bit working registers (the scratchpad) which are fabricated from RAM. The scratchpad can be addressed in two ways:

(a) Direct Addressing: a data field in the ALPS instruction specifies the required register.

(b) Indirect Addressing: a data field in the ALPS instruction specifies that the address will be found in another register. In the case of the LP this can only be the ST register which is used solely as a scratchpad address register for indirect addressing.

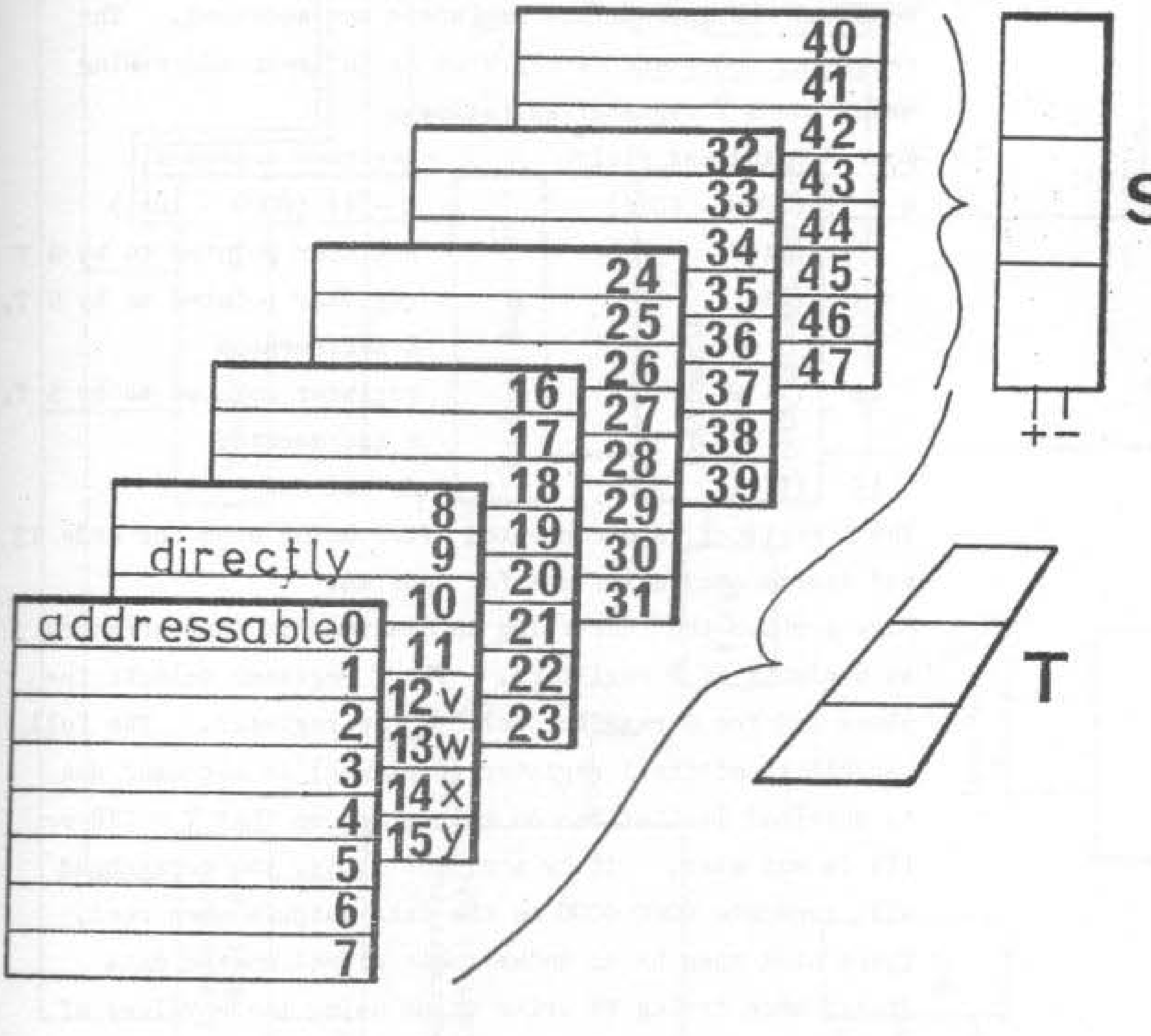


fig.9 L.P. scratchpad addressing scheme

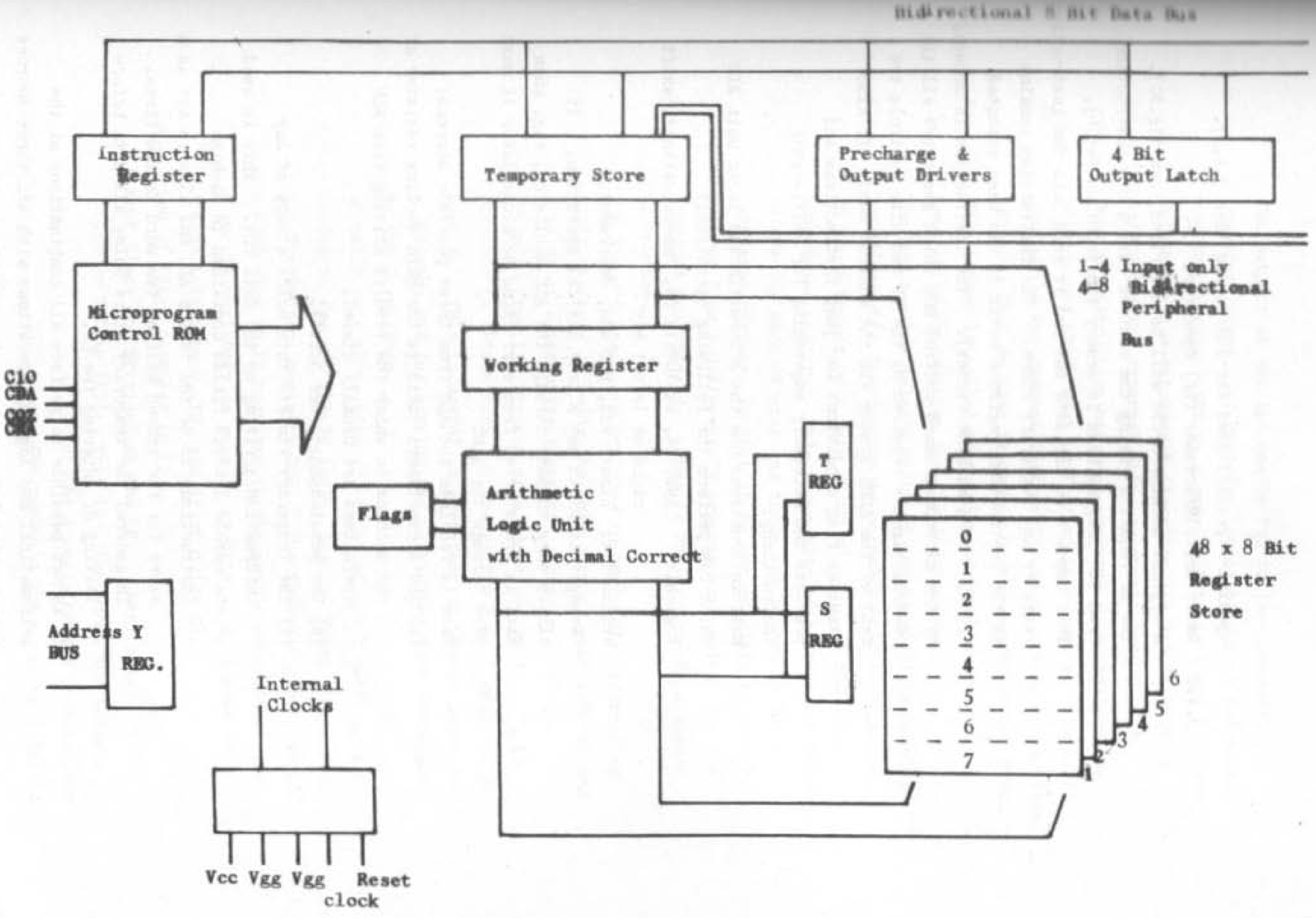
The ST register consists of two three bit registers. T (the most significant three bits) can only be changed using special ALPS instructions which load it with a three bits literal specified in the user's program, (Instruetion: LTS), or from the accumulator (Instruction: SAT). S is a three bit register which can also be loaded from program data (Instruction: LSS) or from the accumulator together with the T register (Instruction: SST). Full details of these instructions will be found in section 5. Register S can also be incremented or decremented. A flag detects when S $\neq 7$ which is used as a jump condition.

A four bit address field is used in the ALPS instruction set to specify the register to be used. Codes 0-11 (0000 - 1011) are interpreted as the direct addressing mode and the appropriate registers are accessed. The remaining codes are interpreted as indirect addressing using the S T register as follows:

Four Bit Address Field	Register Accessed
0 - 11 (0000 - 1011)	0 - 11 (0000 - 1011)
12 (1100)	register pointed to by S T
13 (1101)	register pointed to by S T, S decremented
14 (1110)	register pointed to by S T, S incremented
15 (1111)	do not use

The S register is decremented after being used for code 13 and incremented after use for code 14.

Fig. 9 shows the addressing in diagramatic form arranged as 6 planes of 8 registers. The T register selects the plane and the S register selects the register. The full capability of the T register (8 planes) is not used due to physical limitations on chip size, so that T = 110 or 111 is not used. If by accident it is, the scratchpad will generate 0000 0000 on the data outputs when read. There will then be no undesirable effect on the data stored when trying to write using using these values of T. Note that fig. 9 shows registers 0-15 as directly addressable because special instructions exist to directly address register 12 (also called V), 13 (W), 14 (X) and 15 (Y). Registers 12 and 13 are freely available to the user but 14 and 15 are used for [illegible] INTERNAL BLOCK DIAGRAM OF [illegible]



special registers used for addressing (see 3.1.3).

3.1.2 Basic Logic Processor (LP) Operation

A typical machine cycle will be described (see fig.10). The LP requests (using the control bus) a new instruction from the program held in memory (instruction fetch). This appears on the data bus and is read into the instruction register and temporary store. The instruction remains in the instruction register until it has been executed (1, 2, 3 or 4 machine cycles). The instruction is decoded by the microprogrammed control ROM which generates all the control signals required by the LP and also controls the rest of the ALPS system via the control bus. It also examines flag conditions for jump instructions and provides the necessary sequencing for multi-cycle instructions.

Most instructions use the arithmetic and logic unit ALU which can perform the following operations:

logical:	AND, OR, exclusive OR, 'no-operation', shift right or left 1 bit
arithmetic:	binary add, BCD add, decrement.

'No-operation' is not a true logical operation. It allows data to pass through the ALU unaltered, e.g. when data is transferred from scratchpad to accumulator it must pass through the ALU.

Data is fed into the ALU from three possible sources:

(a) The Accumulator: this is the main working register of the machine in which the results arising from ALU operations are usually stored.

(b) The Scratchpad: (see 3.1.1).

(c) The Temporary Store: this holds a copy of any information arriving on the data bus. This is used to enable literal fields contained in machine instructions to be fed to the ALU and also to act as a store for the second half of two word instructions. In general the temporary store holds all data before leaving or entering the LP.

It is not possible to perform all combinations of the arithmetical and logical operations with all three sources (see section 5 for details of available combinations).

The outputs of the ALU can be fed to the accumulator, temporary store, S.T register, scratchpad or Y extension register (YER). YER is a copy of bits 4, 5 and 6 of the data held in scratchpad Y (location 15) and is used in addressing the external ALPS components (see 3.1.3). When an instruction has been executed a new instruction is required which must be fetched from memory. This is done while the current instruction is on its last execution cycle so that no time is wasted by the memory access time. This scheme is called an overlapped fetch and execute.

3.1.3 Generating Information for the Address Bus.

The address bus selects which of the 64 possible modules is addressed. The program is normally held in pages of 2K words formed of a pair of PM or 1 MIC plus associated memory. Only three bits of address are needed to specify each 2K area and these are stored in the Y extension register so that it is not possible to execute program from a different 2K program area unless the YER contents are changed. This is easily done by a simple subroutine.

The X and Y registers are used in particular addressing instructions. Y holds the full current page address and X holds an alternative address. Program is always executed from the program area addressed by Y. It is possible to obtain data, or change the Q or Z register in page X. This is useful in preparing to change pages in a program, and for storing and retrieving data addressed by the Z register.

Normally only three bits (4, 5, 6) of the Y register are needed to specify the 2K active page. These are held in the Y extension register YER. The full Y register is only needed when it is required to change the contents of the Q (or Z) registers in page Y, as bits 1, 2, 3 must be loaded into bits 9, 10, and 11 of Q (or Z).

3.1.4 Use of flags provided in LP.

Four flags are provided in the LP. There are:

(i) S = 7 flag (S7): This is situated on the outputs of the S register and detects when the contents are equal to 7 (111).

(ii) Carry Flag (CAR): The carry flag detects overflow from bit 8 (MSD) of the ALU. It is used for BCD add (ADR) and when comparing numbers (CMP). It is cleared when the accumulator contents are shifted right one bit (RSA) and set when they are shifted left one bit (LSA).

(iii) Zero Flag (ZER): The zero flag is situated on the outputs of the ALU and detects when 0000 0000 is present.

(iv) Positive Flag (POS): The positive flag detects when the most significant bit (bit 8) of the ALU outputs is 1, i.e. 1xxx xxx. No special significance is attached to bit 8 in the LP system.

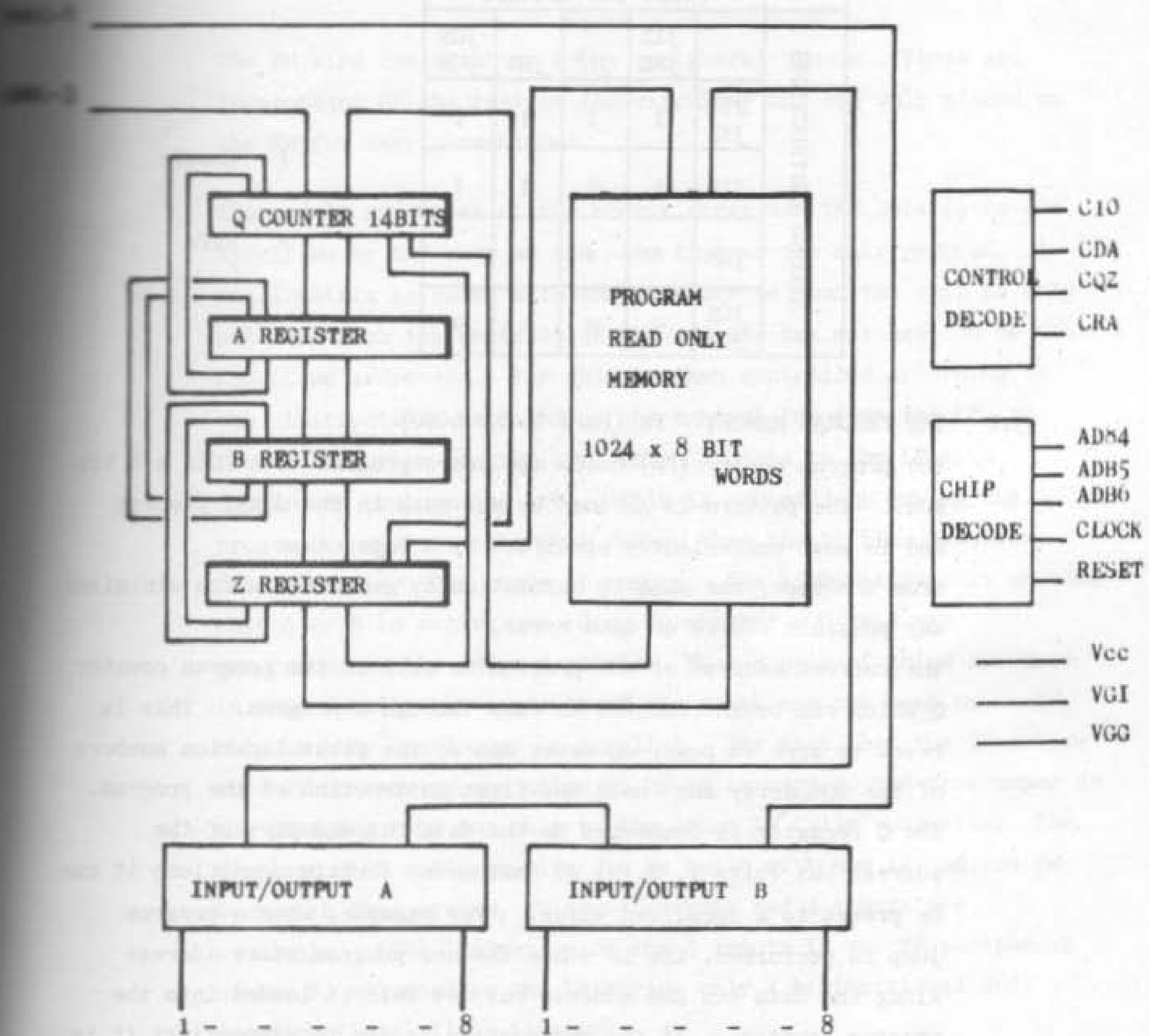
Flags S7 and CAR are static and no special care is needed in their use. Flags ZER and POS are dynamic and it <u>must</u> be remembered that they are set by whatever data is on the ALU outputs. For most instructions that do not explicitly use the ALU functions, the accumulator contents are passed through the ALU and back into the accumulator, and in so doing the ZER and POS flags may be set as follows:

Result equals zero	ZER = 1
Result not equal to zero	ZER = 0
Result positive ($\geqslant 0$)	POS = 1
Result negative (< 0)	POS = 0

If the flags were set by a non-accumulator instruction (i.e. BCD add ADR , decrement DEC or compare CMP) their value may be corrupted during the first cycle of the next instruction. If this is a conditional jump, the positive and negative flags may be changed so that any further conditional jump instructions using POS or ZER may give unexpected results.

The table below shows which jumps may safely follow one another, safe sequences being shown by * and unsafe sequences by 1.

LP 6000 PROGRAM MEMORY. - Fig. 11.



		FIRST INSTRUCTION			
		JIZ JNZ	JIP	JRS	JCS JCN
SECOND INSTRUCTIONS	JIZ JNZ	!	!	!	!
	JIP	!!	!	!	!
	JRS	*	*	*	*
	JCS JCN	*	*	*	*

!	Unsafe
*	Safe

3.2 THE PROGRAM MEMORY - PM (Part No.LP 6000)

The program memory (PM) holds the users program in a 1024 x 8 bit ROM. The pattern is defined by one mask in the GIANT process and is most conveniently specified by a paper tape.
From the tape, the mask is automatically generated which minimises any possible source of data error.
The current address of the program is held in the program counter Q which can be incremented to step through a program. This is reset to zero at power-on-reset and so the first location numbers of the ROM array must hold the first instruction of the program. The Q register is connected to the data bus and part of the address bus (bits 1, 2, 3) so that under certain conditions it can be preset to a specified value. For example, when a program jump is performed, the LP sends the new program start address along the data bus and address bus and this is loaded into the program counter. If the instruction is 'go to subroutine' it is necessary to store the old value of the program counter. For this reason Q is the top register of a stack of four registers which allows nesting of subroutines to a depth of three, i.e. the user may program a subroutine of a subroutine of a subroutine called by the main program. The remaining registers RA, RB and Z are not reset at power on zero. This is of no concern for correctly written programs but may cause confusion if the stack is popped before any value has been pushed into RA, RB or Z. Z can be preset to a given value in the same way as the Q counter. Care must be taken to ensure that this is not done when the maximum depth of subroutining is being used because Z being the bottom of the stack, holds the final return address.

The PM also contains two 8 bit peripheral busses.. These are independant of the rest of the circuitry and are only placed on the PM for user convenience.

The module addresses of the memory array and the interfaces are specified by the user at the same time as the main program. A small matrix is coded with these values so that the chip is only selected when the contents of the address bus matches one of the specified addresses. The chip is then controlled according to the instructions received on the control bus from the LP, to form a 2K active program area which appears to the LP as a continuous programming area.. This is achieved by making the program counter 11 bits long rather than the 10 bits necessary for addressing 1024 words of memory. The 11th bit selects whether chip A or B is active according to whether a 0 or 2 was programmed in the address matrix. When a pair of chips are used in this fashion the contents of the stack are the same on each chip as they are loaded in parallel. The fact that the 2K active area is spreadover two chips is of no concern to the programmer as the changing from one chip to the other is fully automatic. The simplest ALPS configuration can be built with one LP and one PM. Even with this system, 20 input/outputs and 4 inputs are available. (The limitation of the 4 inputs is due to peripheral number 63 situated on the LP having only 4 bidirectional and 4 undirectional lines (inputs).

3.3 MEMORY INTERFACE - Part No. LP 1000

The ALPS system normally uses the PM for program storage. However, for program development this may be inconvenient to use as the masked read only memory is obviously impractical for debugging purposes. The memory interface chip (MIC) enables any form of read/write memory to be used with the ALPS system, e.g. RAM, PROM, core, diode matrix. In operation it is almost identical to the four register stack provided on the PM except that the address bits are brought out of the package via TTL compatible outputs. It can also address up to 2K of externald memory compared with 1K for PM. A maximum of 8 MICs can be used in the ALPS system.

When used with a normal read/write memory an IOB chip will be necessary to buffer the data (see below) & Fig. LP0001.

There is no reason why the MIC plus external memory should not be used instead of the PM but the larger package count and increased cost means that this approach is only suitable for small production runs or where it is likely that the program may require changing.

The address of the MIC is determined by hardwiring external pins as follows.

Connections

PAD 6	PAD 5	PAD 4	MIC Module Address Range
Vgg	Vgg	Vgg	0 - 7
Vgg	Vgg	Vcc	8 - 15
Vgg	Vcc	Vgg	16 - 23
Vgg	Vcc	Vcc	24 - 31
Vcc	Vgg	Vgg	32 - 39
Vcc	Vgg	Vcc	40 - 47
Vcc	Vcc	Vgg	48 - 55
Vcc	Vcc	Vcc	56 - 63

A chip select is provided to inhibit the MIC operation. The address bus, control bus, PAD inputs and chip enable are all TTL compatible.

3.4 INPUT/OUTPUT BUFFER - Part No. LP1010

The basic ALPS system provides 24 input/outputs in the basic two chip (LP + PM) system.

As the program becomes longer more PMs will be added to the system, so that input/outputs will increase with program complexity. Many simple systems often need only the simplest of programs but many I/Os and for these applications the IOB has been introduced to allow economical expansion of I/O capability without paying for unnecessary .extra program space. It consists of two 8 bit I/O parts of the standard open-ended ALPS pattern (see fig.6). The addresses are selected by three pins which are connected to Vgg or Vcc. Addresses are only selectable in adjacent pairs in the range 48-63 as follows:

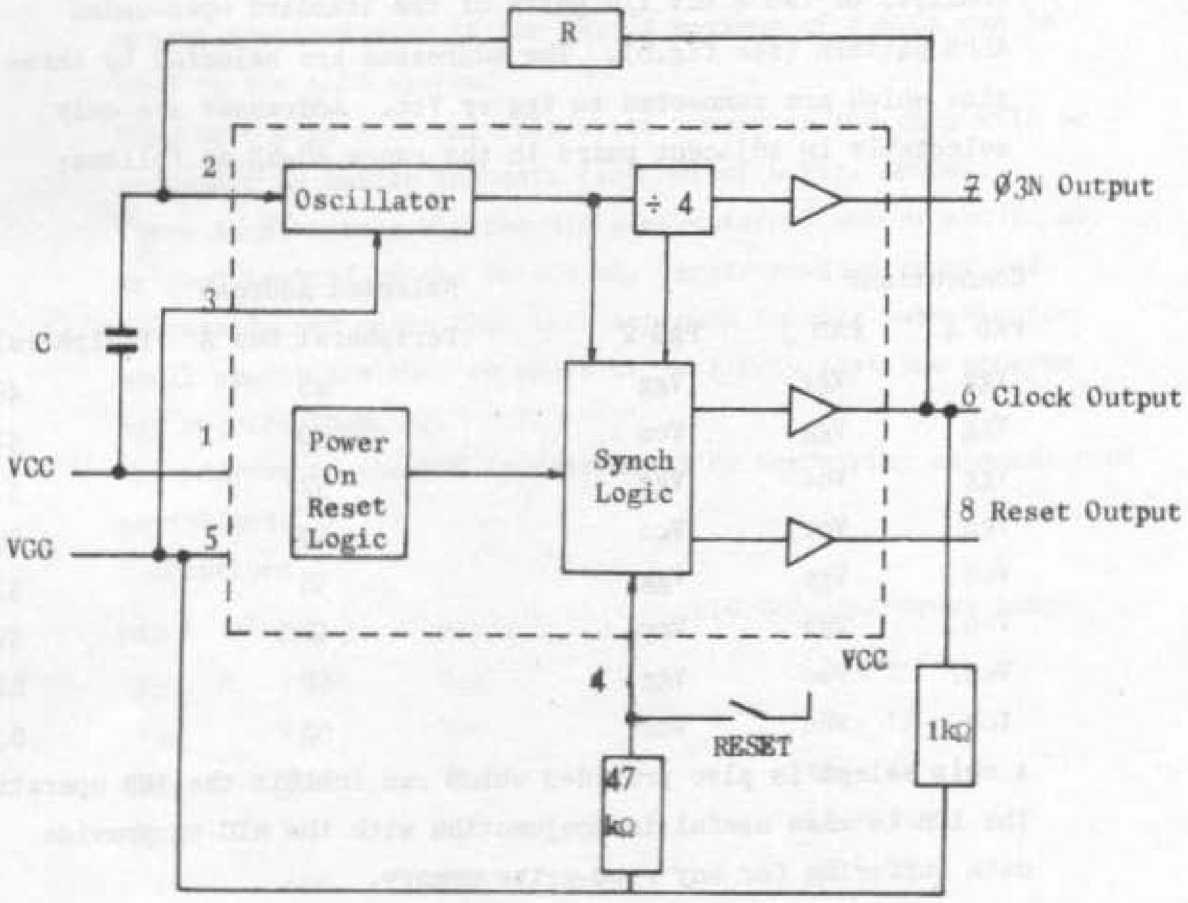
Connections			Selected Address	
PAD 4	PAD 3	PAD 2	Peripheral Bus A	Peripheral Bus B
Vgg	Vgg	Vgg	48	49
Vgg	Vgg	Vcc	50	51
Vgg	Vcc	Vgg	52	53
Vgg	Vcc	Vcc	54	55
Vcc	Vgg	Vgg	56	57
Vcc	Vgg	Vcc	58	59
Vcc	Vcc	Vgg	60	61
Vcc	Vcc	Vcc	62	63

A chip select is also provided which can inhibit the IOB operation. The IOB is also useful in conjunction with the MIC to provide data buffering for any read-write memory.

The address bus, control bus, PAD inputs and chip enable are all TTL compatible.

3.5 CLOCK GENERATOR - Part No. LP 1030

The ALPS system needs only a simple high amplitude clock and a power on reset synchronized with the clock. This can most easily be provided using the clock generator (CG). A block diagram of this is shown below. The circuit generates the necessary clock and reset signals for the system. The Reset and Synchronizing Inputs are directly compatible with TTL/DTL systems.



Frequency 500-800KHz $0 \leq R \leq 100K$ $0 \leq C \leq 100pF$

Using the Clock-Generator in Self-Oscillating Mode Fig. 12

For the Clock Output to follow the Synchronizing Input in the slaved mode, the Timing Input should be tied to a logic '0'.

The ALPS Microprocessor System requires a reset signal that is synchronized to the system clock, and at Power On the system must have a minimum of two clock periods of reset to force all component circuits into their initial state. Logic on the LP 1030 monitors the power supply lines to the system and generates on the Reset Output (pin 8) a signal giving this minimum of two clock periods of reset for the system. The negative edge of this signal, which enables the system, is correctly synchronized to the system clock and the Ø3N signal. This latter condition ensures that the Reset Output is only enabled at one particular period of a machine cycle of the system, one period of Ø3N defining one machine cycle. If, however, during the power up period the Reset Input is at a logic '1', the Reset Output will remain in its positive active state until this Reset Input is taken to a logic '0'. The Reset Output after this input change will switch to its inactive negative state synchronised to the system clock and Ø3N signal for the reason previously described.

The LP 1030 monitors the Reset Input during that period when Ø3N is positive and the system clock is negative. If between these periods the Reset Input has changed from a logic '0' to a logic '1' then the Reset Output will change to its positive state on the positive edge of the Ø3N subsequent to the LP 1030 monitoring the change of input.

For Reset Input changes from a logic '1' to a logic '0', the Reset Output will change to its negative state on the first positive edge of the system clock occuring during the positive period of Ø3N after this input change.

The Reset Output has an internal pull down resistor to Vgg, nominal value 2.5 KOhm.

The Ø3N Output also has an internal pull down resistor to Vgg, nominally 3.5 KOhm. The Ø3N signal is of importance in the ALPS Microprocessor System because the Data Bus of the system is true only during that period when Ø3N is positive. Its availability assists system debugging.

The Clock and Reset Outputs of the LP 1030 can drive typically a system comprising of six circuits without additional circuitry. For additional drive requirements the Clock Output will require bipolar buffering, however. The Reset and Ø3N Output drive capability can be increased by connecting 1KOhm pull down resistors to Vgg from these outputs.

MAXIMUM RATINGS *

All pins with respect to Vcc	-20V to 0.3V	* Exceeding these ratings could cause permanent damage to the device. Functional operation is not implied - operating conditions are specified below.
Storage Temperature	-55°C to $+150^{\circ}$C	
Operating Temperature	0°C to $+75^{\circ}$C	

ELECTRICAL CHARACTERISTICS (LP 1030 - see Data Sheet for others).

Vcc = +5V ± 0.25V; Vgi = GND (substrate at Vcc); Vgg = -12V ± 1V.

Parameter	Symbol	Min.	Typ	Max.	Unit	Condition
Supply Voltage	Vcc	+4.75		+5.25	V	
	Vgg	-13		-11	V	
Reset Input						
Logic '1'	Vrii	Vcc -1.5			V	
Logic '0'	Vrio			0.8	V	
Synchronizing Input						
Logic '1'	Vsii	Vcc -1.5			V	
Logic '0'	Vsio			0.8	V	
All Outputs						
Logic '1'	Voi	Vcc -1			V	Sinking 20mA
Logic '0'	Voo			-10	V	Vgg = -12V Vcc = +5V
Input Leakage	Iil			1	uA	Vin = Vgg at 25°C
Capacitance on Inputs	Ci			10	pF	
Clock Output						
Width	tø1	450		700	nS	
Rise Time	tør			200	nS	1KOhm pull down driving 60pF
Fall Time	tøf			200	nS	As tør
Ø3N Output						
Rise Time	t3r			300	nS	60pF load
Fall Time	t3f			500	nS	60pF load
Propogation Delay						
logic '0' to '1'	tø1			300	nS	
logic '1' to '0'	tø0			300	nS	
Reset Output						
Delay Time						
logic '1' to '0'	trf	0	400	700	nS	
logic '0' to '1'	trr		400		nS	
Reset Input						
Let up Time	tørl	300			nS	
	tørt	100			nS	
Synchronising Input Frequency	f			800	KHz	
Pulse Width High		450			nS	
Pulse Width Low		450			nS	

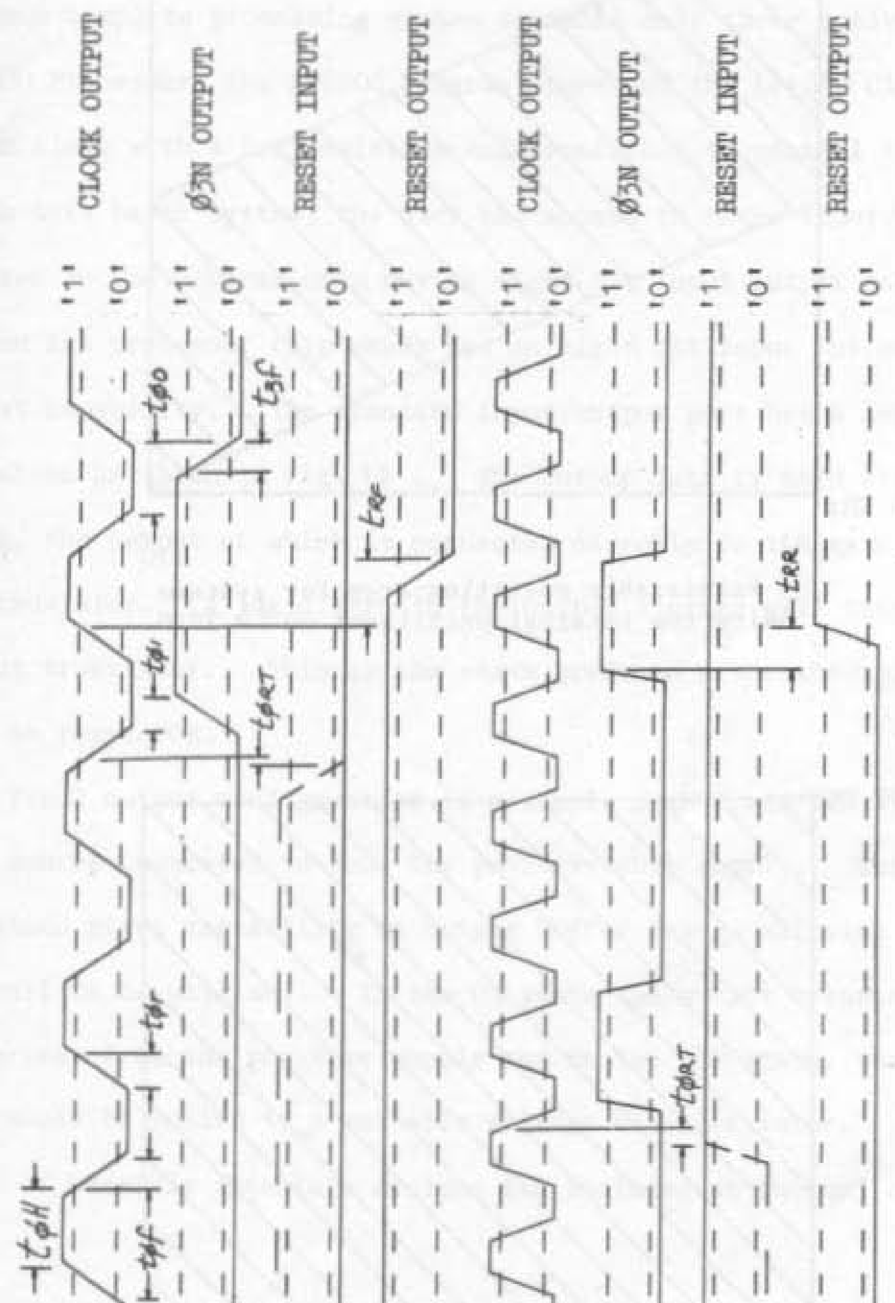
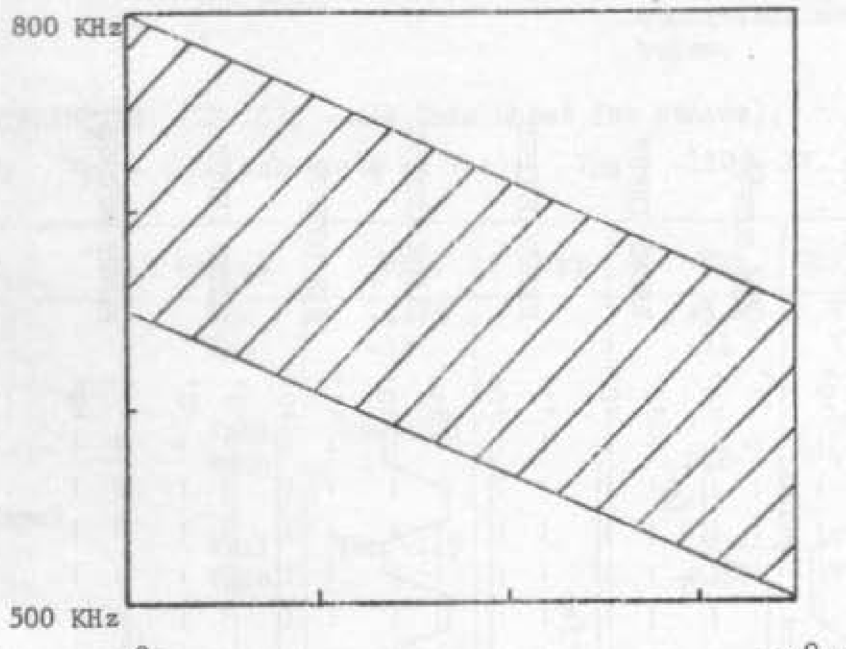
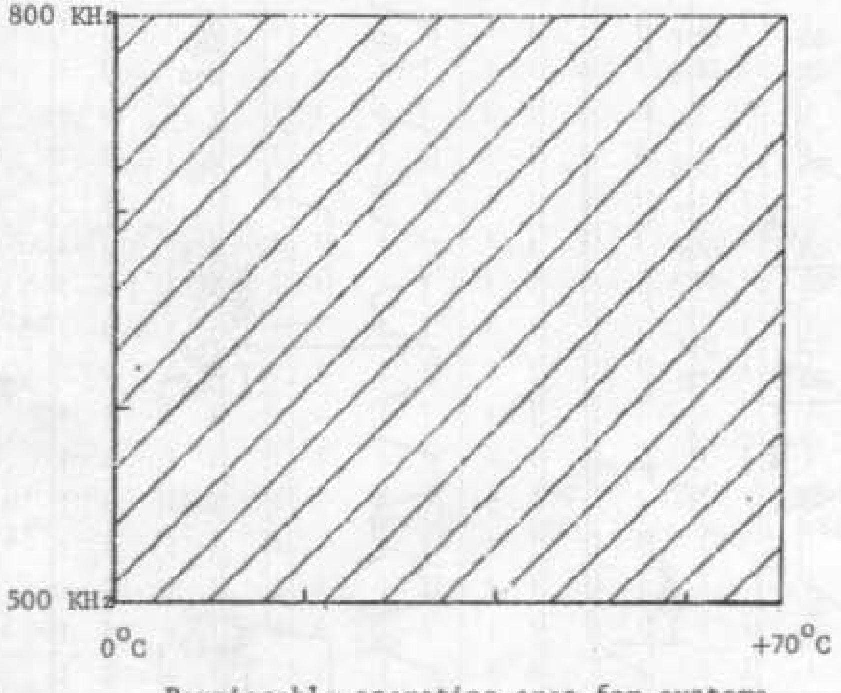


FIG.13. TIMING DIAGRAM LP1030



Permissable operating area for systems using the internal oscillator on LP 1030.



Permissable operating area for systems using external oscillator to LP 1030.

4.0 INTERFACE CONSIDERATIONS

The Advanced Logic Processing System ALPS has been designed to a strict philosophy of being very simple to implement in a users system. The minimum complete processing system contains only three active chips, the LP8000 Processor, the LP6000 Program Store and the LP1030 Clock Generator along with a few resistors and capacitors to control the clock.

With this basic system, the user has access to three input/output groups, two on the program chip having eight bit input output capability and one on the processor chip which has an eight bit input but only a four bit output capability. The standard input/output port has a logic configuration as shown in fig. 14 . The output data is held static by a flip flop, the output of which is connected directly to the gate of the output transistor. A logic zero in the output storage will turn OFF the output transistor. This is the state produced after the application of power on reset POR.

The final output configuration is a simple open drain MOS transistor with its source connected to Vcc, the positive chip supply. This configuration gives versatility in output buffer design allowing any voltage rail to be utilised. In the ON state the output transistor will source current from the positive supply and in the OFF state, the output normally would be pulled to a suitable voltage via a resistor. A selection of possible interface designs are included at the end of this section.

When the user wishes to use the input facility, it must be remembered that the output register is directly coupled to the output transistor and the output word would appear wired-or on any input data. It is common in several programs to use the input port to read a previous output word but if this is not the intention, the output register should be cleared to all zeros before any input is requested.

/...2

The input/output ports contained on the individual I/OB chip are slightly different from those contained in the other chips, in that the open drain output transistors can source twice the output current for the same Vds voltage drop. This feature can simplify several interface designs, and for example it allows direct coupling to a TTL input with only a single pull down resistor.

The inputs are directly compatible with several other logic families including TTL and CMOS. When used with simple switches, an antibounce facility can be incorporated in the program if switch bounce was likely to be a problem.

/...3

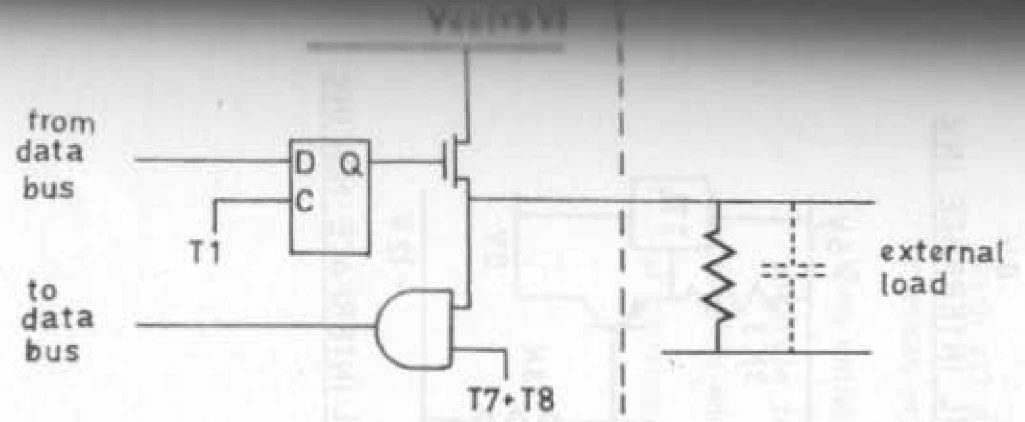
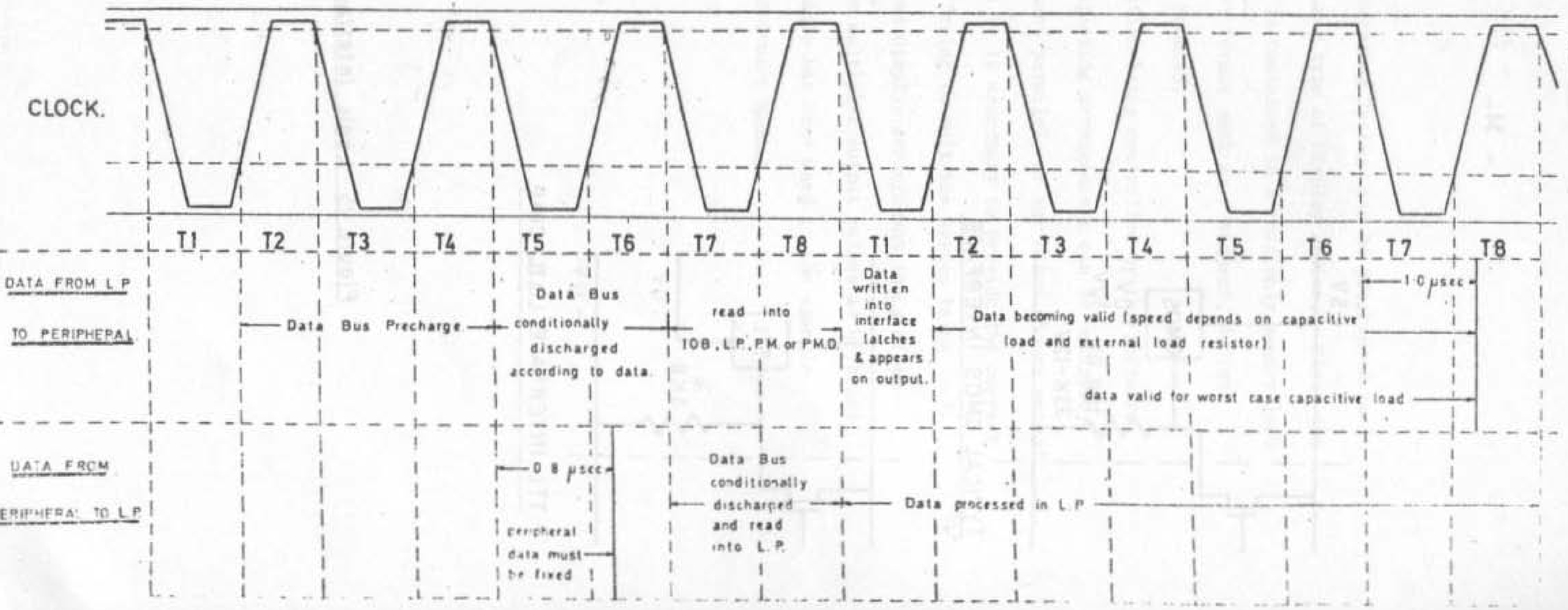


Figure 14
Input Output Configuration


CLOCK.
T1 T2 T3 T4 T5 T6 T7 T8 T1 T2 T3 T4 T5 T6 T7 T8
DATA FROM L.P TO PERIPHERAL
Data Bus Precharge
Data Bus conditionally discharged according to data.
read into IOB, L.P, P.M or P.M.D.
Data written into interface latches & appears on output.
Data becoming valid (speed depends on capacitive load and external load resistor)
1·0 µsec
data valid for worst case capacitive load
DATA FROM PERIPHERAL TO L.P
0·8 µsec
peripheral data must be fixed
Data Bus conditionally discharged and read into L.P.
Data processed in L.P

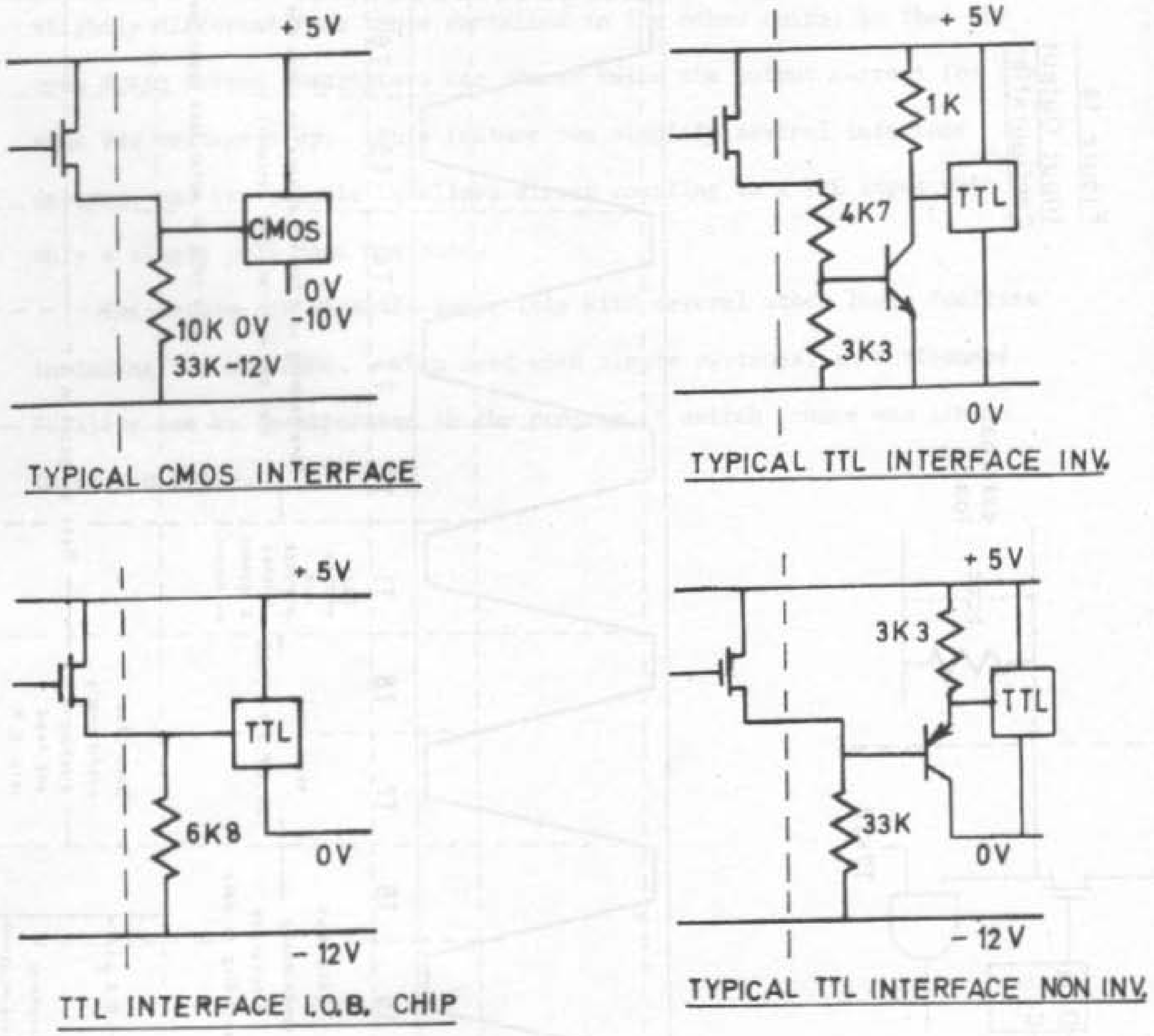


Figure 15 Logic Interfaces

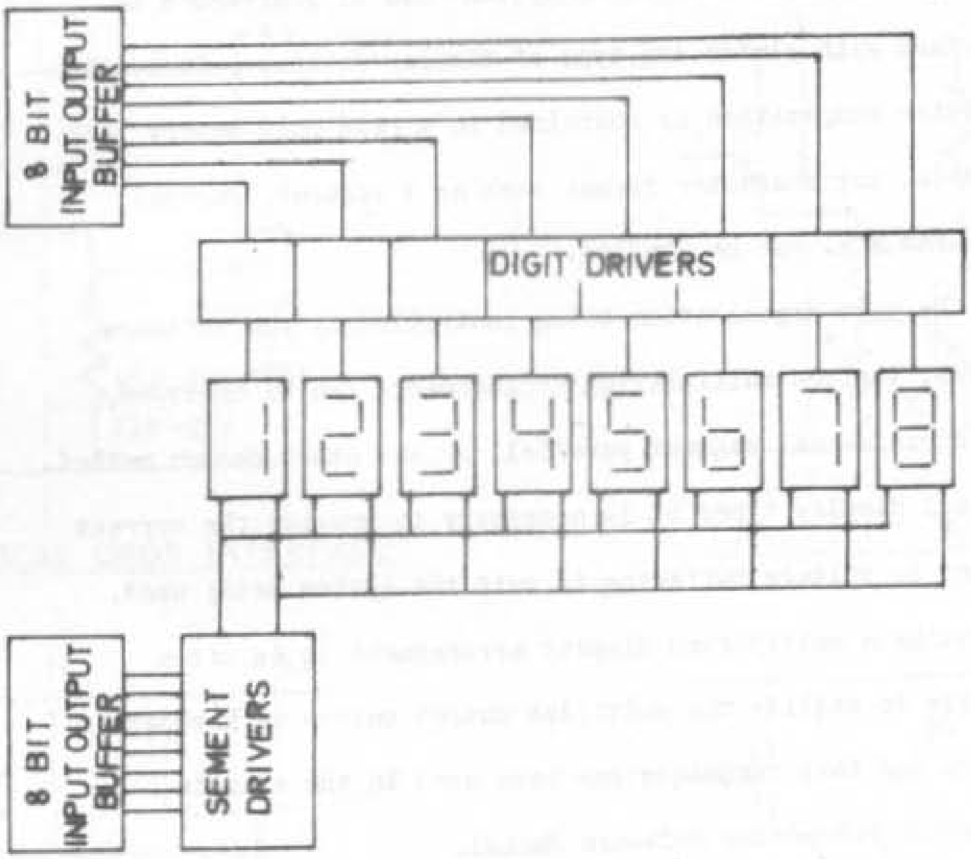
4.1 Display Interfacing

The Advanced Logic Processing System can be programmed to interface with almost any type of display device. Since the character composition is contained in a read only memory look up table, any character format such as 7 segment, decimal starburst etc. can be handled.

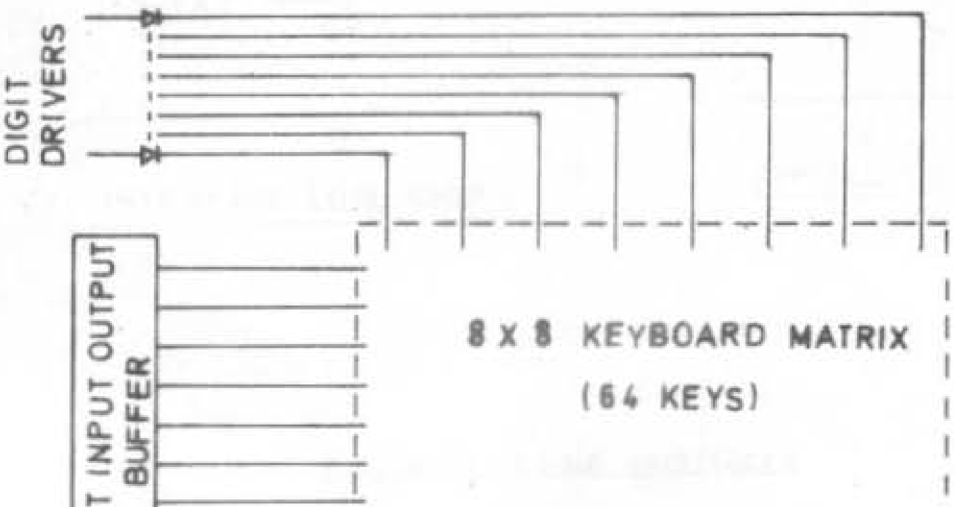
With the data organisation being controlled by the software program, various multiplexing arrangements can be employed, i.e. digit serial segment parallel, or any other chosen method.

With all display types it is necessary to provide the correct current or voltage buffering to suit the system being used.

When using a multiplexed display arrangement it is often possible to utilise the multiplex output pulses as keyboard strobes and this technique has been used in the example of section 7.0 of the Software Manual.

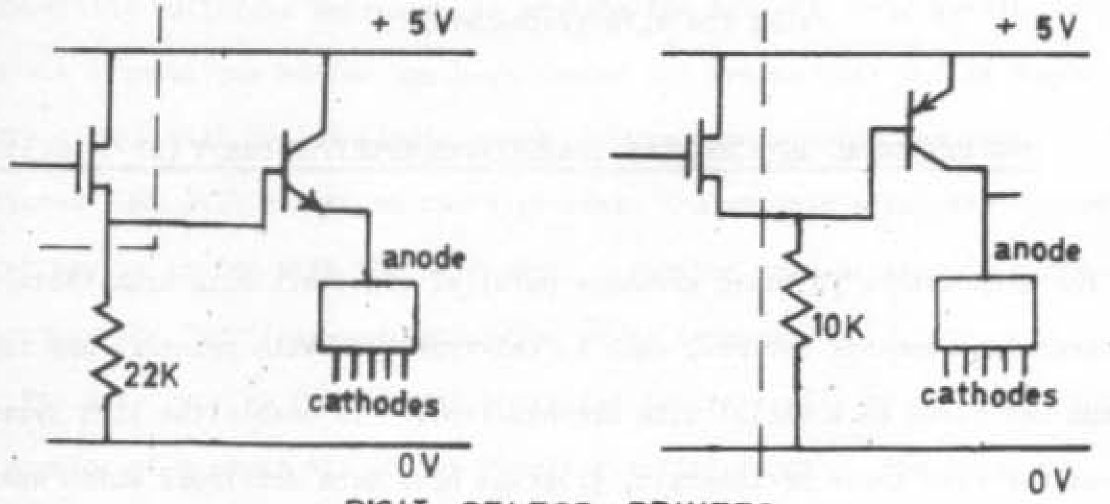


— EIGHT DIGIT DISPLAY FORMAT —

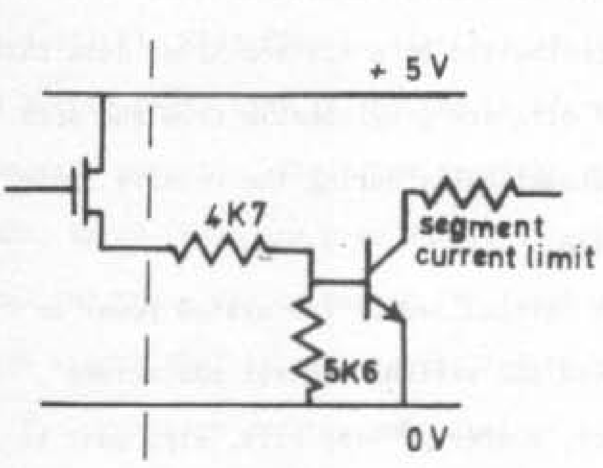


KEYBOARD MATRIX FORMAT

Figure 16



DIGIT SELECT DRIVERS
for COMMON ANODE DISPLAYS



SEGMENT DRIVER

Figure 17 Display Interfaces

4.2 Using the ALPS system with

THE UNIVERSAL ASYNCHRONOUS TRANSMITTER/RECEIVER UAR/T (AY-5-1013)

The ALPS system is based around a parallel eight bit data architecture, but several peripheral devices, such as teletypes, certain printers and tape systems are based on a serial data organisation. To enable the ALPS system to converse with these peripherals, programs have been developed which uses the UAR/T device to provide a serial data converter. The UAR/T device contains all the logic necessary to provide an asynchronous serial data interface which can transmit or receive variously formatted data. The UAR/T provides a variable baud rate controlled by a sixteen times data rate clock and the word format and control bits are programmable from the ALPS system. The data word is checked automatically during the receive cycle and various error conditions are tested.

In operation the UAR/T device is initialised by the system power on reset and before operation is initiated the various control characters which program word length, parity mode, number of stop bits, etc. must be set, and a Control Strobe (CS), must be applied. In most applications the word format will be fixed and the control word and control strobe will be hard wired. A level is sufficient to operate the control strobe, a complete pulse is not required.

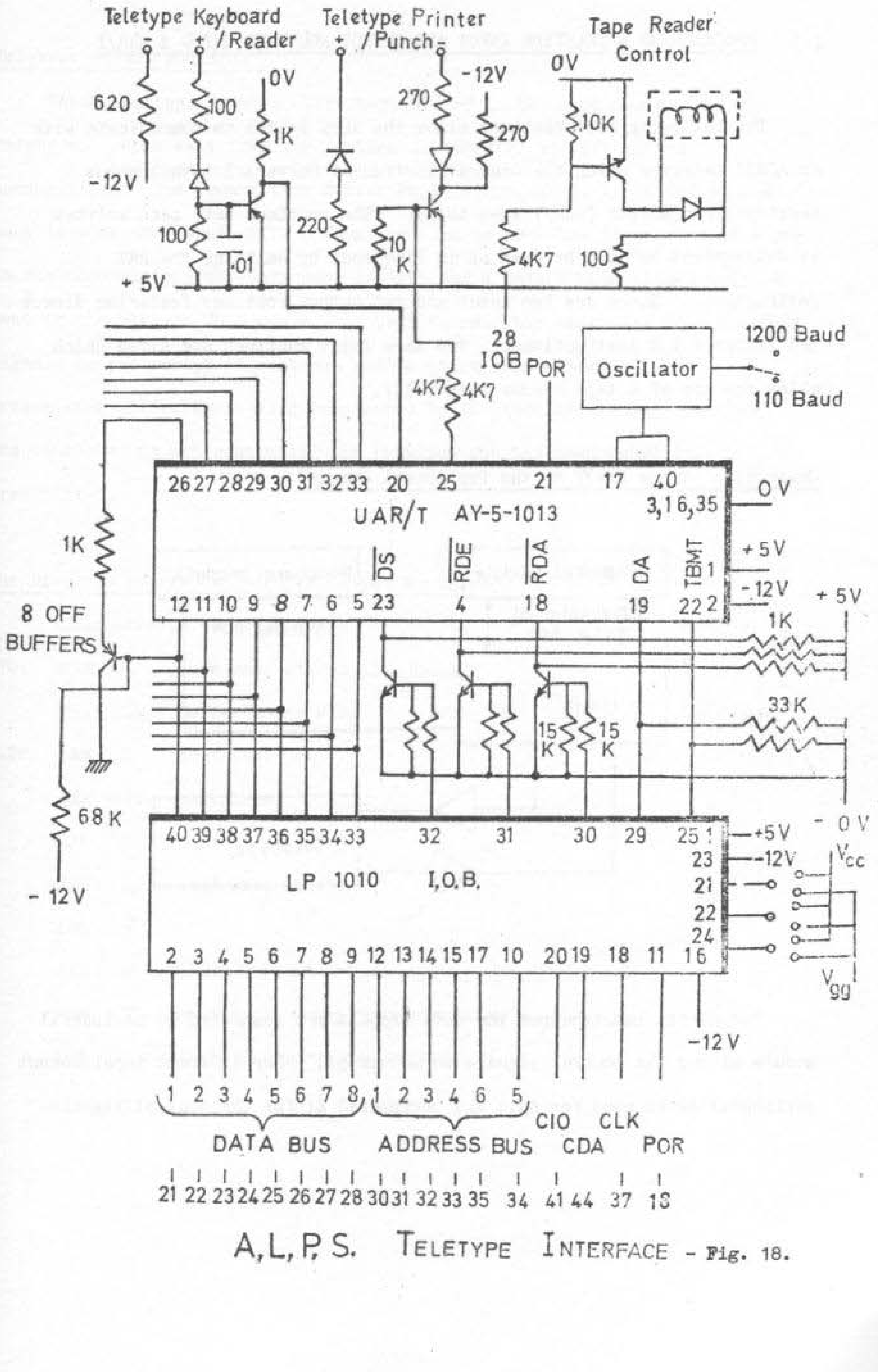
With all control conditions complete the UAR/T device may be used to transmit or receive and in full duplex if necessary. If transmission is required, the processor must set up the parallel data word on the appropriate input/output buffer which will be connected to the parallel inputs of the transmitter buffer register. The processor must now output a negative going Data Strobe signal ($\overline{DS}$) which loads the transmitter buffer during the negative period and initiates the data transmission from the positive edge.

An inverting buffer is necessary to provide the correct logic levels. When the transmitter buffer has been loaded the Transmitter Buffer Empty (TBMT) signal will go low (logic zero). Once transmission has been initiated TBMT will return to the high state and at this point the transmitter buffer may be loaded with the next word. Further transmission is controlled by testing the TBMT flag and outputting $\overline{DS}$ as necessary.

The data receive function is activated automatically by the UAR/T upon the sensing of a start bit on the receiver serial input. The receiver section will complete the entry and checking function on the incoming serial data stream and upon reaching the centre of the first stop bit it will present the data in parallel to the receiver buffer register. The Data Available flip flop is also set at this time and providing the Status Word Enable ($\overline{SWE}$) line is low the DA signal will be gated onto its respective output. The other receiver status words Parity Error (PE) Framing Error (FE), and Over Run (OR), are also controlled by the $\overline{SWE}$ signal and these can be tested for level if required. The transmitter status signal TBMT is also controlled by the $\overline{SWE}$ input.

The processor program must include a polling routine which regularly inspects the DA output , also operating the $\overline{SWE}$ input if it is not hard wired. When DA is found to be at a high (logic one) level the processor must then operate the Received Data Enable ($\overline{RDE}$) line which gates the parallel data from the receiver buffer onto the tri-state ouput lines. The program must then provide an internal data store sequence to load the accumulator from the peripheral. This will input the received data to the processor. After the data has been read the DA flip flop must be reset by pulsing the Reset Data Available ($\overline{RDA}$) line. The UAR/T system will then be ready to transfer the next received data. The transmitter and receiver sections of the UAR/T are double buffered and data may be in transfer on the communication line while other data is being input or output from the processor.

An external oscillator is necessary for operating the clock of the transmitter and receiver registers. The clock inputs are TTL compatible and the frequency must have a stability of better than four percent and an absolute value of sixteen times the sum of the total word length and baud rate.

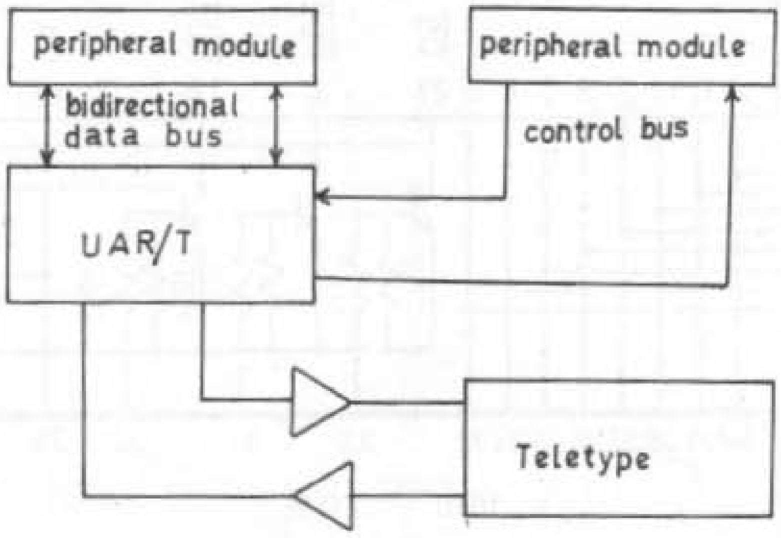


A,L,P,S. TELETYPE INTERFACE - Fig. 18.

4.3 PROGRAMMING A TELETYPE INPUT AND OUTPUT ROUTINES USING A UAR/T

The following four routines allow the ALPS system to communicate with an ASR33 teletype using the General Instrument Universal Asynchronous Receiver/Transmitter (UAR/T AY-5-1013). The routines have been written as subroutines but can be used as in line code by omitting the RET instruction. There are two input and two output routines featuring direct and indirect I/O instructions. Two more input routines are given which allow the use of a tape reader step relay.

Connection of the UAR/T to the Peripheral Channel



For direct input/output the data signals are connected to peripheral module 61 and the control signals to module 62. For indirect input/output peripheral 40 is used for data and peripheral 41 for the control signals.

Teletype Output Routines

These routines print a character, stored in the accumulator, on the teletype. Upon exit from the routine a character is left in the accumulator. The Transmitter Buffer Empty flags, TBMT, is tested in the busy loop at the label WAIT. When the flag becomes set (appearing as a one in the accumulator) the busy loop is left and a DATA STROBE signal ($\overline{DS}$) is sent to the UART. This causes the UART to read the character from the data signals on the peripheral latches and to start transmission. The Transmitter Buffer Empty Flag is cleared by the Data Strobe Signal, thus one character is not sent until the previous one has been completely transmitted.

The Directly Addresses Teletype Output Routine

```
        Character is in Accumulator
TTO:    SAM   5    save character on latches
        Test UAR/T Buffer Empty Flag
WAIT:   LAM   6    test TBMT
        JIP Wait   Busy loop
        LAS   1
        SAM   6    send Data Strobe
        LAS   Ø
        SAM   6    clear Data Strobe
        LAM   5    reload character to accumulator
        RET        optional subroutine return
```

Indirectly Addressed Teletype Output Routine

This routine acts in exactly the same way as the directly addressed routine. The difference is that instead of using LAM and SAM instructions it uses LIM and SIM instructions with the X register loaded with a module address. The previous contents of the X register are stored in register 11 and the character, held in the accumulator on entry to the routine, is saved in register 10. The busy loop is at the label WAIT as before:

```
TTO:    SAR   10    save character in temporary register
        LAX   ,     save register X
        SAR   11    in temporary register
        LAL   40    set up data I/O address
        SAX   ,     store it in X
        LAR   10    get character -
        SIM   ,     store on latches
        LAL   41    set up -
        SAX   ,     control signals peripheral
WAIT:   LIM   ,     is character in UART?
        JIP   Wait  busy loop
        LAS   1
        SIM   ,     send Data Strobe
        LAS   Ø
        SIM   ,     clear Data Strobe
        LAR   11
        SAX   ,     restore X register
        LAR   10    reload character to accumulator
        RET         optional return
```

Teletype Input Routine

As with teletype output there is a flag which must be tested before proceeding with the I/O. This is the DATA AVAILABLE flag of the UART and is set when a character has been received by the UART. It is cleared by the signal RESET DATA AVAILABLE ($\overline{\text{RDA}}$) to allow another character to be received. To open the tristate receiver buffer of the UART, thus allowing the character to be read, the signal RECEIVED DATA ENABLE ($\overline{\text{RDE}}$) must be applied.

The Directly Addressed Teletype Input Routine

```
TTT: LAM    6    Test Data Available
     LSN
     JIP   TTI
     LAS    2
     SAM    6    Send RDE
     LAS    0
     SAM    5    clear data latches
     LAM    5    read character and
     SAM    5    save on latches
     LAS    4    send RDA and clear RDE
     SAM    6
     LAS    Ø    clear RDA
     SAM    6
     LAM    5    restore character to
     RET         accumulator
```

Indirectly Addressed Teletype Input Routine

As the two teletype output routines differed in their addressing modes so do the two input routines. The indirect routine similarly using LIM and SIM instructions.

```
TTI:  LAX        save X register
      SAR   11   in temporary store
      LAL   41   set address of -
      SAX    ,   control signals
Busy: LIM    ,   read in control signals
      LSN
      JIP   Busy test Data Available
      LAS    2
      SIM    ,   send RDE
      LAL   40
      SAX    ,   data peripheral
      LAS    0
      SIM    ,   clear latches
      LIM    ,   read  character and
      SAR   10   store in register 10
      LAL   41
      SAX
      LAS    4   send RDA and clear RDE
      SIM
      LAS    Ø   clear RDA
      SIM
      LAR   11
      SAX    ,   restore X register
      LAR   10   reload character
      RET
```

<u>Teletype Input Routines Featuring Reader Stepping</u>

The reader step relay is connected to bit five of the control signals peripheral and is actuated during the busy loop of the input routine. Thus the directly addressed input routine becomes:

```
TTI:    LAL   /10    set bit five
        SAM    6     actuate reader
Busy:   LAM    6
        LSN
        JIP   Busy
        LAS    2     send $\overline{RDE}$ and release relay
        SAM    6
        LAS    0
        SAM    5     clear latches
        LAM    5     read character and
        SAM    5     save on latches
        LAS    4
        SAM    6     send $\overline{RDA}$ and clear $\overline{RDE}$
        LAS    ∅
        SAM    6     clear $\overline{RDA}$
        LAM    5     restore character
        RET
```

and the indirectly address routine.

```
TTI:    LAX
        SAR   11
        LAL   41
        SAX
        LAL   41
        SAX
        LAL   /10
        SIM          actuate relay
```

```
Busy:  LIM
       LSN
       JIP   Busy   test data available
       LAS    2
       SIM          send $\overline{RDE}$ and release relay
       LAL   40
       SAX
       LAS    Ø
       SIM          clear latches
       LIM          read character and
       SAR   10     save in register 10
       LAL   41
       SAX
       LAS    4     send $\overline{RDA}$ and clear $\overline{RDE}$
       SIM
       LAS    Ø     clear $\overline{RDA}$
       SIM
       LAR   11
       SAX          restore X register
       LAR   10     reload character
       RET
```

5. THE INSTRUCTION SET

The ALPS instruction set consists of 48 instructions which can be 1, 2, 3 or 4 machine cycles. Most of the commonly used instructions are 1 or 2 machine cycles long. As the system has an overlapped fetch and execute, the run time of any program can be calculated by adding up the number of cycles of each instruction and multiplying by the machine cycle time (5uS min.)

The instruction set is presented in different formats as follows:

5.1 A description of the instruction set showing some uses of each instruction.

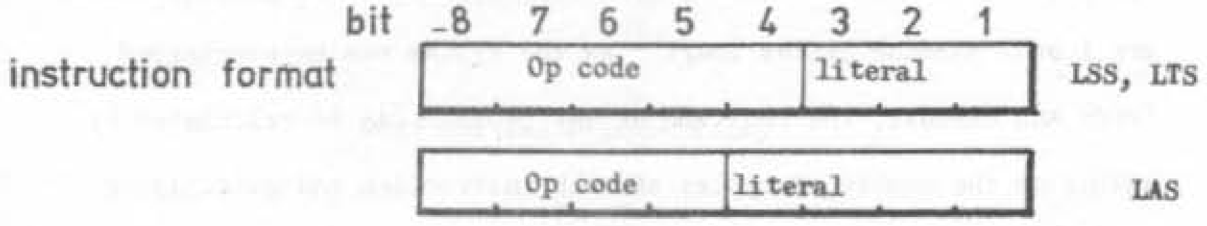
5.2 A list of the instruction set grouped by function. This is useful when writing programs and looking for an instruction to perform a given function.

5.3 A list of the instruction set grouped by op code order. This is used when checking the contents of memory or checking program coding.

5.4 A list of the instruction set showing a detailed breakdown of data flow and the state of the address bus, data bus and control bus during the instruction execution. This is useful when debugging a system and also shows the precise manner in which the instructions work.

5.1 A Description of the Instruction Set

SHORT LITERAL INSTRUCTIONS



OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
1111 ****	LAS	load accumulator with short literal	1
0010 1***	LSS	load S with short literal	1
0011 1***	LTS	load T with short literal	1

The instruction LSS loads the S register with the three bit literal which appears in the instruction. The contents of the accumulator are unaffected. The instruction LTS loads the T register in a similar fashion. These two instructions are used to set up a start address before addressing the L.P. scratchpad indirectly.

The instruction LAS loads the accumulator with a four bit short literal (bits 1-4). The most significant bits 5-8 are filled with zeros. This is useful for introducing single B.C.D. numbers or any other 4 bits of data into the system in a single cycle.

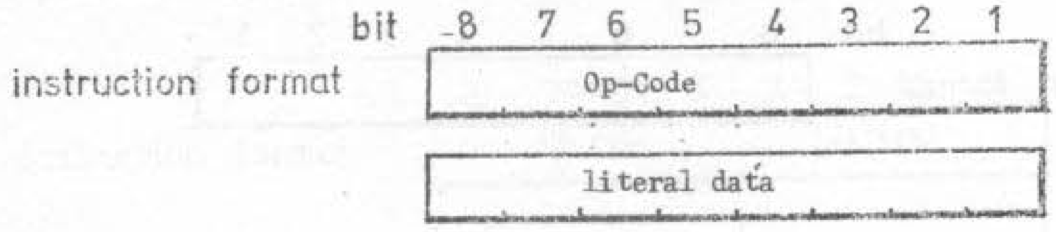
REGISTERS S AND T

bit	8	7	6	5	4	3	2	1
instruction format	op code							

OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
0000 0010	SAT	store accumulator in register T	1
0000 0011	SST	store accumulator in registers S and T	1

The registers S and T (used indirectly for accessing the RAM registers) can be loaded by the short literal instructions LSS and LTS or from the accumulator using the instructions SAT and SST. When the value required in the T register is arrived at by computation (it is not known until the program is run and thus cannot be held as a short literal) it can be loaded using the SAT instruction.
The S register can be loaded at the same time by using the SST instructions. An example of the use of the SST instruction might be when the RAM registers are being used as a lookup table (storing a list of values). If the S and T registers are loaded with data received from an input the values read from RAM register by indirect access will be a function of the input signal (i.e. a conversion from ASCII to another code.)

LONG LITERAL INSTRUCTIONS



OP CODE	MNE-MONIC	INSTRUCTION	CYCLES
0000 0100	LAL	load accumulator with long literal	2
0000 0101	ALL	'AND' long literal with accumulator	2
0000 1100	EOL	'EXCLUSIVE OR' long literal with accumulator	2
0000 1101	ORL	'OR' long literal with accumulator	2
0000 1110	ALA	binary add, long literal to accumulator	2
0000 1111	CMP	compare long literal with accumulator	2

These six instructions fetch a long (8 bit) literal from the next word of program memory and perform arithmetic (add, load, compare) or logical operations (AND, EXCLUSIVE OR, OR) with the literal on the contents of the accumulator. The logical operations may be used to manipulate or examine individual bits of a word for example, to test whether bit 5 of a word in the accumulator is set the following instructions could be used.

```
ALL /10  (/10 is the literal data in hexadecimal
               i.e. 00010000)
JNZ SET
```

The ALL instruction will perform a logical AND of the literal with the accumulator effectively setting all the bits except bit five to zero. If bit five was set before the ALL instruction the JNZ (jump if non-zero) instruction will be satisfied and the program will jump to the label set. If however, bit five was clear before the test, the accumulator would be zero at the JNZ instruction. Thus the jump condition would not be satisfied and the program would continue at the next instruction.

The compare instruction CMP is used to test the value of a variable (a counter or an arithmetic result) against an eight bit literal. The contents of the accumulator and the literal are added together in the Arithmetic and Logical Unit and the status flags (CARRY, POSITIVE,ZERO) set or cleared according to the result. The contents of the accumulator remain unchanged. For example the test for the character carriage return (ASCII value 13) in a teletype input routine the code below could be used (with the character in the accumulator)

```
        CMP CR

        JIZ CARRET
```

Because the compare instruction uses the addition facility of the A.L.U. the two's complement of the character is inserted in the literal. Thus for the character carriage return (0000 1101) the literal would be 11110011. Adding carriage return to the literal gives:-

```
          0000 1101
          1111 0011
        1 ---------
          0000 0000

    carry   accumulator
```

Thus the result is zero.

To demonstrate using the compare instruction to test the result of an arithmetic operation the following instructions test if the contents of the accumulator lie between zero and nine.

```
         JIP OK
         JMP ERROR      number less than zero

  OK:    CMP -9
         JIP ERROR      number greater than nine
```

The compare instruction adds the centre of the accumulator and the literal -9 and sets the positive flag if the accumulator contained a number greater than nine. Thus the JIP ERROR instruction will detect the erroneous condition.

ACCUMULATOR AND REGISTERS V,W,X,Y

	bit 8	7	6	5	4	3	2	1
instruction format	op code							

OP CODE	MNE-MONIC	INSTRUCTION	CYCLES
0000 1000	LAV	load accumulator from register V	1
0000 1001	LAW	load accumulator from register W	1
0000 1010	LAX	load accumulator from register X	1
0000 1011	LAY	load accumulator from register Y	1
0001 1000	SAV	store accumulator in register V	1
0001 1001	SAW	store accumulator in register W	1
0001 1010	SAX	store accumulator in register X	1
0001 1011	SAY	store accumulator in register Y	1

The four registers V,W,X,Y cannot be accessed directly by the LAR or SAR instructions because the op codes for LAR or SAR on registers 12,13, 14 and 15 are given special meaning by the processor. Thus eight special instructions are set aside for loading and reading these registers. The action of the instructions is identical to the action of the LAR and SAR instructions. The registers X and Y have a particular significance as module addressing registers. See ADDRESS CONTROL INSTRUCTIONS (page 57).

REGISTER ACCESS INSTRUCTIONS

	bit 8	7	6	5	4	3	2	1
instruction format	op code				argument			

OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
1000 ####	LAR	load accumulator from register	1
1001 ####	SAR	store accumulator in register	1
1010 ####	BAD	binary add register to accumulator	1
1101 ####	DEC	Decrement register by one	1
1110 ####	ADR	BCD add accumulator to register	2
1011 ####	AND	Logical 'AND' register with accumulator	1
1100 ####	EOR	'Exclusive OR' register with accumulator	1

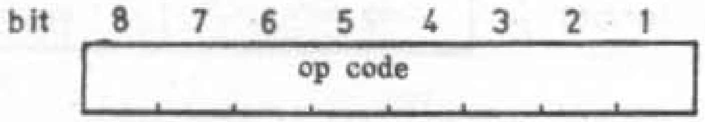
The above instructions are used to manipulate the contents of the accumulator and the ram registers. They have a four bit argument (0-15). Register access is direct if the argument lies between 0 and 11 but indirect for arguments 12, 13 and 14. The action is indefined for argument 15. The Decrement instruction (DEC) is used to maintain a counter or pointer using a register. For example a loop which must be entered 7 times:-

```
                LAS 6
                SAR REG
        LOOP      |
                  |
                  |
                DEC REG
                JIP LOOP
```

Decimal addition is accomplished using the BCD add instruction ADR. This two cycle instruction adds the contents of the accumulator to the register during the first cycle and performs a BCD correction during the second cycle. The decimal carry is set or cleared by the result of the addition and is used by the next decimal addition. The contents of the accumulator and register are treated as two four bit binary coded decimal digits. Using indirect auto increment (or auto decrement) register access two sixteen digit decimal numbers can be added together. Repeated addition and shifting can give multiplication. The logical operations of AND and Exclusive OR are provided to enable bit manipulation of the contents of the registers.

SHIFT INSTRUCTIONS

instruction format



OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
0001 1100	LSA	Shift accumulator 1 bit left	1
0001 1101	RSA	Shift accumulator 1 bit right	1
0001 1110	LSN	Shift accumulator 4 bits left	1
0001 1111	RSN	Shift accumulator 4 bits right	1

These instructions are self explanatory. They are used to manipulate the bits of a word, to multiply or divide a two digit decimal number by ten (LSN and RSN). The single bit shift instructions LSA and RSA set and clear the carry flag respectively.

ADDRESS CONTROL INSTRUCTIONS

	bit 8	7	6	5	4	3	2	1
instruction format	op code							

OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
0001 0010	SZX	Store accumulator in Z on module X	3
0001 0011	SZY	Store accumulator in Z on module Y	3
0001 0110	SQX	Store accumulator in Q on module X	3
0001 0111	SQY	Store accumulator in Q on module Y	3

To increase the addressing capabilities of the processor two of the internal RAM registers X and Y are designated ADDRESS CONTROL REGISTERS. The Y register contents define the program area, whereas the X register defines the data area. The four address control instructions store an eleven bit address (formed from bits 1-3 of the appropriate register and the eight bits of the accumulator) in the Q or Z register of the module addressed by X or Y register. Thus to store data in a location, the X register must be loaded to point to the module of the location. The address of the location within the module must then be loaded into the accumulator. Execution of an SZX instruction will load the Z register with the location address. The data can then be stored (or fetched) using a DATA TRANSFER instruction SIX (or LIX). Similarly, the Q register can be loaded using an SQX to provide a start address of a routine in another program area. The instructions SAX, SQX and SAY are used to produce a change of program area as demonstrated in the examples.

Examples

Fetching data from a module

```
        LAL     MODATA
        SAX                     set up module address
        LAL     ADDATA
        SZX                     set up intra-module address
        LIX                     fetch data
```

MODATA is the module number of the location and ADDATA is the address of the location within the module.

Storing data in a module

```
        LAL     MODATA
        SAX
        LAL     ADDATA
        SZX

        load data to accumulator

        SIX
```

Changing program area

```
        LAL     NEWAREA
        SAX                     set up module address
        LAL     STARTADDR
        SQX                     set up start address
        LAX
        SAY                     change program to new area
        SAY                     dummy instruction
```

The SAY instruction changes the program over to the new area but since the instryction execution and fetch are overlapped, it has to be followed by a dummy instruction which will not affect the operation of the changeover. Another SAY instruction is a good choice of dummy instruction.

DATA TRANSFER INSTRUCTION

	bit 8	7	6	5	4	3	2	1
instruction format	op code							

OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
0000 0110	LIX	Load accumulator with (Z), module X	4
0000 0111	LIY	Load accumulator with (Z), module Y	4
0000 0010	SIX	Store accumulator in (Z), module X	3
0000 0110	LIM	Load accumulator from peripheral, module X	4
0000 0010	SIM	Store accumulator in peripheral, module X	3

The instructions LIX, LIY and SIX are used to fetch and store data in a module indirectly via the Z register of that module. The use must be preceded by an SZX or SZY instruction as described in the examples for the ADDRESS CONTROL INSTRUCTIONS. It is possible to store in and load from the data area (X register - SIX and LIX) but only possible to load from the program area (Y - register - LIY). This is because the program area is usually Read Only Memory.

The instructions LIM and SIM are used to fetch and store data from I/O peripheral devices, The particular I/O module is specified by the lower 6 bits of the X register, The use of the LIM and SIM instructions must be preceeded by an SAX instruction to set up the required module address.

JUMP INSTRUCTIONS

	bit 8	7	6	5	4	3	2	1
instruction format	Op code					argument		
	argument							

OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
0100 0*** **** ****	JMP	Unconditional jump	3
0100 1*** **** ****	JIZ	Jump when result is zero	3/2
0101 0*** **** ****	JNZ	Jump when result is non-zero	3/2
0101 1*** **** ****	JIP	Jump when result is positive	3/2
0110 0*** **** ****	JRS	Jump if register S is not equal to seven	3/2
0110 1*** **** ****	JCS	Jump if carry set	3/2
0111 0*** **** ****	JCN	Jump if carry not set	3/2

The destination address of the jump instruction is generated from the three bit argument in the actual instruction and the eight bit argument in the next word. This generates an eleven bit address which allows a range for the jump instruction of 2K.

The conditional jump instructions JIZ, JNZ, JIP, JRS, JCS and JCN test the status flags. If the appropriate condition is satisfied the program behaves as if an unconditional jump JMP had been encountered. If the condition is not satisfied the program continues with the next sequential instruction. The unconditional jump instruction takes 3 cycles, the first to fetch and decode the instruction, the second to fetch the nextword from memory and the third to load the progam counter with the new address. An unsuccessful conditional jump goes through the first two cycles but does not load the program counter. Thus the time taken for a conditional jump is described as 3/2 machine cycles.

INPUT/OUTPUT INSTRUCTIONS

instruction format

bit	8	7	6	5	4	3	2	1
	op code					argument		

OP. CODE	MNE-MONIC	INSTRUCTION	CYCLES
0010 0***	LAM	Load accumulator from module	2
0011 0***	SAM	Store accumulator in module	3
		module = (*** +56)	

Input or output to a peripheral is a lengthy time-consuming process when the LIM or SIM instructions are used. To increase the speed of input/output the LAM and SAM instructions are provided. These have a three bit argument (0-7) allowing eight modules to be selected. Since modules 0-7 in a system will usually be ROM an offset of 56 is added to the argument to move the directly addressed I/O's up to 56-63.

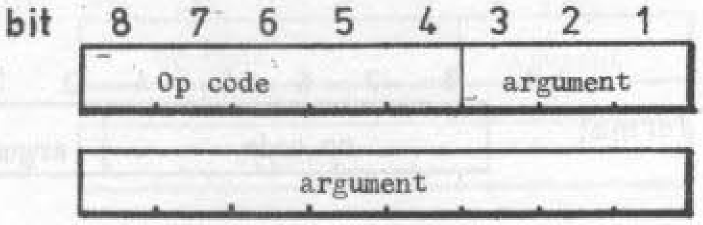
A comparison of the use of SIM and SAM instructions to put the data held in register REG onto peripheral 57.

```
LAL  57                      LAR REG
SAX     set up module
        address              SAM 1        send data
LAR  REG
SIM     send data
```

The indirect addressing code needs 5 words of memory and takes 7 cycles to execute whereas the directly accessed system needs 2 words and takes 4 cycles.

SUBROUTINE INSTRUCTIONS

instruction format



OP CODE	MNE-MONIC	INSTRUCTION	CYCLES
0111 1***	GOS	Go to subroutine (push stack)	3
**** ****			
0000 0000	RET	Return from subroutine (pull stack)	2

The 'Go to subroutine' instruction GOS generates an eleven bit address in the same manner as the jump instructions. However, instead of this address overwriting the contents of the program counter the old address is saved in another register (pushed onto the stack) and then the new address is loaded into the program counter. Up to three old addresses can be saved on the stack before overflow. Return from subroutine causes the last address stacked to be "popped" from the stack and loaded into the program counter. In this way a piece of code can be entered by a jump-like instruction from anywhere in the program with the added advantage of being able to return to the exit point in the main program.

ALPS INSTRUCTION SET SUMMARY BY INSTRUCTION GROUPING

Instructions for Loading and Storing Registers

MNEMONICS English	CODE	FUNCTION	CYCLES	WORDS	STATUS FLAGS Positive	STATUS FLAGS Zero	STATUS FLAGS Carry	See note
Accumulator & registers (direct addressing):								
LAL	0000 0100 **** ****	Load accumulator with long literal	2	2	*	*	-	
LAS	1111 ****	Load accumulator with short literal	1	1	*	*	-	
LAR	1000 ****	Load accumulator from register	1	1	*	*	-	5
SAR	1001 ****	Store accumulator in register	1	1	*	*	-	5
Accumulator & registers (indirect addressing):								
LSS	0010 1***	Load S with short literal	1	1	*	*	-	6
LTS	0011 1***	Load T with short literal	1	1	*	*	-	6
SST	0000 0011	Store accumulator in reg. S.T.	1	1	*	*	-	6
SAT	0000 0001	Store accumulator in reg. T (bits 1, 2, 3)	1	1	*	*	-	6
Accumulator & address control registers:								
LAV	0000 1000	Load accumulator from reg. V	1	1	*	*	-	
LAW	0000 1001	Load accumulator from reg. W	1	1	*	*	-	
LAX	0000 1010	Load accumulator from reg. X	1	1	*	*	-	
LAY	0000 1011	Load accumulator from reg. Y	1	1	*	*	-	
SAV	0001 1000	Store accumulator in V	1	1	*	*	-	
SAW	0001 1001	Store accumulator in W	1	1	*	*	-	
SAX	0001 1010	Store accumulator in X	1	1	*	*	-	
SAY	0001 1011	Store accumulator in Y	1	1	*	*	-	
SZX	0001 0010	Store accumulator in Z on Module X	3	1	*	*	-	
SZY	0001 0011	Store accumulator in Z on Module Y	3	1	*	*	-	
SQX	0001 0110	Store accumulator in Q on Module X	3	1	*	*	-	
SQY	0001 0111	Store accumulator in Q on Module Y	3	1	*	*	-	
Input/								

ALPS INSTRUCTION SET SUMMARY BY INSTRUCTION GROUPING

Instruction for Logical, Addition and Shift Operations

MNEMONICS English	CODE	FUNCTION	CYCLES	WORDS	STATUS FLAGS Positive	Zero	Carry	See Note
Logical Operations:								
ALL	0000 0101 **** ****	Logical AND with accumulator long literal	2	2	*	*	-	
AND	1011 ****	Logical AND with register and accumulator (Binary)	1	1	*	*	-	5
ORL	0000 1101 **** ****	OR with accumulator long literal	2	2	*	*	-	
EOL	0000 1100 **** ****	Exclusive OR with accumulator long literal	2	2	*	*	-	
EOR	1100 ****	Exclusive OR with register and accumulator	1	1	*	*	-	5
Arithmetic operations:								
BAD	1010 ****	Add register to accumulator (Binary)	1	1	*	*	-	5
DEC	1101 ****	Decrement register by 1	1	1	*	*	-	5
ADR	1110 ****	BCD add accumulator to register (Decimal)	2	1	*	*	*	2, 5
ALA	0000 1110 **** ****	ADD literal to accumulator (carry not set)	2	2	*	*	-	
CMP	0000 1111 **** ****	Compare accumulator with literal (sets status flip flops)	2	2	*	*	*	2
Shift Operations:								
LSA	0001 1100	Shift accumulator 1 bit left	1	1	*	*	Set	2
RSA	0001 1101	Shift accumulator 1 bit right	1	1	*	*	Cleared	2
LSN	0001 1110	Shift accumulator 4 bits left	1	1	*	*	-	
RSN	0001 1111	Shift accumulator 4 bits right	1	1	*	*	-	

KEY: * = Flag affected by operation

- = Flag not affected

Instructions for Loadind and Storing Registers (contd.)

MNEMONICS		CODE	FUNCTION	CYCLES	WORDS	STATUS FLAGS			See note
English	German					Positive	Zero	Carry	
Input/Output		(direct addressing): -							
LAM		0010 0***	Load accumulator from module (***+56)	2	1	*	*	-	
SAM		0011 0***	Accumulator to module (***+56)	3	1	*	*	-	
Input/Output		(indirect addressing):							
LIM		0000 0110	Load accumulator from peripheral, module X	4	1	*	*	-	
SIM		0000 0010	Store accumulator in peripheral, module X	3	1	*	*	-	
Data Transfer		(indirect addressing):							
LIX		0000 0110	Load accumulator with (Z) from Module X	4	1	*	*	-	
LIY		0000 0111	Load accumulator with (Z) from Module Y	4	1	*	*	-	
SIX		0000 0010	Store accumulator in (Z), Module X	3	1	*	*	-	

KEY: * = Flag affected by operation
- = Flag not affected

ALPS INSTRUCTION SET SUMMARY BY INSTRUCTION GROUPING

Instructions for Transferring Control

MNEMONICS English	CODE	FUNCTION	CYCLES	WORDS	STATUS FLAGS Positive	Zero	Carry	See Note
Jump within a module:								
JMP	0100 0*** **** ****	Unconditional Jump	3	2	↕	↕	-	1
JIZ	0100 1*** **** ****	Jump when result = 0	3/2	2	↕	↕	-	1
JNZ	0101 0*** **** ****	Jump when result ≠ 0	3/2	2	↕	↕	-	1
JIP	0101 1*** **** ****	Jump when result is positive	3/2	2	↕	↕	-	1
JRS	0110 0*** **** ****	Jump when register S ≠ 7	3/2	2	↕	↕	-	1
JCS	0110 1*** **** ****	Jump if carry flip flop set	3/2	2	↕	↕	-	1
JCN	0111 0*** **** ****	Jump if carry flip flop not set	3/2	2	↕	↕	-	1
Subroutine instructions:								
GOS	0111 1*** **** ****	Go to subroutine (Push Stack)	3	2	↕	↕	-	1
RET	0000 0000	Return from subroutine (Pull Stack)	2	1	↕	↕	-	1
Processor control and undefined instructions:								
HLT	0001 0000	Halt (Simulator instruction only, not implemented in processor)	1	1	-	-	-	3
NOP	0001 0001	Undefined operation	1	1	↕	↕	↕	4
NOP	0001 0100	Undefined operation	1	1	↕	↕	↕	4
NOP	0001 0101	Undefined operation	1	1	↕	↕	↕	4

KEY: * = Flag affected by operation; - = Flag not affected; ↕ = Flag become undefined

5.3

ALPS INSTRUCTION SET SUMMARY IN OP-CODE ORDER

OP-CODE	MNEMONIC		FUNCTION	CYCLES	WORDS	STATUS FLAGS			See Note
	English	German				Positive	Zero	Carry	
0000 0000	RET		Return from Subroutine (Pull Stack)	2	1	!	!	-	1
0000 0001	SAT		Store accumulator in register T	1	1	*	*	-	6
0000 0010	SIX		Store accumulator in (Z), module X	3	1	*	*	-	
0000 0010	SIM		Store accumulator in peripheral, Module X						
0000 0011	SST		Store accumulator in register ST	1	1	*	*	-	6
0000 0100 **** ****	LAL literal		Load accumulator with long literal	2	2	*	*	-	
0000 0101 **** ****	ALL literal		Logical and accumulator with long literal	2	2	*	*	-	
0000 0110	LIX		Load accumulator with (Z), module X	4	1	*	*	-	
0000 0110	LIM		Load accumulator from peripheral module X						
0000 0111	LIY		Load accumulator with (Z), module Y	4	1	*	*	-	
0000 1000	LAV		Load accumulator from register V	1	1	*	*	-	
0000 1001	LAW		Load accumulator from register W	1	1	*	*	-	
0000 1010	LAX		Load accumulator from register X	1	1	*	*	-	
0000 1011	LAY		Load accumulator from register Y	1	1	*	*	-	
0000 1100 **** ****	EOL literal		Exclusive OR accumulator, with long literal	2	2	*	*	-	
0000 1101 **** ****	ORL literal		OR, accumulator with long literal	2	2	*	*	-	
0000 1110 **** ****	ALA literal		Add long literal to accumulator	2	2	*	*	-	
0000 1111 **** ****	CMP literal		Compare accumulator with long literal	2	2	*	*	*	2

KEY: * = Flag affected by operation; - = Flag not affected; ! = Flag becomes undefined

ALPS INSTRUCTION SET SUMMARY IN OP-CODE ORDER

OP-CODE		MNEMONIC		FUNCTION	CYCLES	WORDS	STATUS FLAGS			See Note
		English	German				Positive	Zero	Carry	
0001	0000	HLT		Halt processor (simulator instruction only)	1	1	I	I	I	3
0001	0001	NOP		Undefined operation	1	1	I	I	I	4
0001	0010	SZX		Store accumulator in Z , module X	3	1	*	*	-	
0001	0011	SZY		Store accumulator in Z , module Y	3	1	*	*	-	
0001	0100	NOP		Undefined operation	1	1	I	I	I	4
0001	0101	NOP		Undefined operation	1	1	I	I	I	4
0001	0110	SQX		Store accumulator in Q , on module X	3	1	*	*	-	
0001	0111	SQY		Store accumulator in Q , on module Y	3	1	*	*	-	
0001	1000	SAV		Store accumulator in register V	1	1	*	*	-	
0001	1001	SAW		Store accumulator in register W	1	1	*	*	-	
0001	1010	SAX		Store accumulator in register X	1	1	*	*	-	
0001	1011	SAY		Store accumulator in register Y	1	1	*	*	-	
0001	1100	LSA		Shift accumulator 1 bit left	1	1	*	*	Set	2
0001	1101	RSA		Shift accumulator 1 bit right	1	1	*	*	Cleared	2
0001	1110	LSN		Shift accumulator 4 bits left	1	1	*	*	-	
0001	1111	RSN		Shift accumulator 4 bits right	1	1	*	*	-	

KEY: * = Flag affected by operation; - = Flag not affected; I = Flag becomes undefined

ALPS INSTRUCTION SET SUMMARY OP-CODE ORDER

OP-CODE	MNEMONIC		FUNCTION	CYCLES	WORDS	STATUS FLAGS			See Note
	English	German				Positive	Zero	Carry	
0010 0***	LAM		Load accumulator from module (***+56)	2	1	*	*	-	
0010 1***	LSS		Load S with short literal (0-7)	1	1	*	*	-	6
0011 0***	SAM		Store accumulator in module (***+56)	3	1	*	*	-	
0011 1***	LTS		Load T with short literal (0-7)	1	1	*	*	-	6
0100 0*** **** ****	JMP		Unconditional Jump (sets Q = 11 bit literal address)	3	2	I	I	-	1
0100 1*** **** ****	JIZ		Jump if result equals zero (sets Q = 11 bit literal address)	3/2	2	I	I	-	1
0101 0*** **** ****	JNZ		Jump if result non-zero (set Q = 11 bit literal address)	3/2	2	I	I	-	1
0101 1*** **** ****	JIP		Jump if positive (set Q = 11 bit literal address)	3/2	2	I	I	-	1
0110 0*** **** ****	JRS		Jump if register S $\neq$ 7 (set Q = 11 bit literal address)	3/2	2	I	I	-	1
0110 1*** **** ****	JCS		Jump if carry set (set Q = 11 bit literal address)	3/2	2	I	I	-	1

KEY: * = Flag affected by operation; - = Flag not affected; I = Flag becomes undefined

ALPS INSTRUCTION SET SUMMARY OP-CODE ORDER

OP-CODE	MNEMONIC English	MNEMONIC German	FUNCTION	CYCLES	WORDS	STATUS FLAGS Positive	STATUS FLAGS Zero	STATUS FLAGS Carry	See Note
0111 0*** **** ****	JCN		Jump if carry not set (set Q = 11 bit literal address)	3/2	2	I	I	-	1
0111 1*** **** ****	GOS		Go to subroutine (push stack) (set Q = 11 bit literal address)	3/2	2	I	I	-	1
1000 ****	LAR		Load accumulator from register (****)	1	1	*	*	-	5
1001 ****	SAR		Save accumulator in register (****)	1	1	*	*	-	5
1010 ****	BAD		Add register (****) to accumulator, binary	1	1	*	*	-	5
1011 ****	AND		Logical AND, register (****) with accumulator	1	1	*	*	-	5
1100 ****	EOR		Exclusive OR, register (****) with accumulator	1	1	*	*	-	5
1101 ****	DEC		Decrement register (****) by one	1	1	*	*	-	5
1110 ****	ADR		BCD add accumulator to register (****)	2	1	*	*	*	2, 5
1111 ****	LAS		Load accumulator with short literal	1	1	*	*	-	

KEY: * = Flag affected by operation; - = Flag not affected; I = Flag becomes undefined

5.4 NOTES ON INSTRUCTION SET

Note 1 - Condition of Status Flags with Conditional Jump Instructions

During a multicycle instruction the contents of the accumulator are passed through the Arithmetic and Logic Unit (ALU), and may be operated upon, depending on the instruction. When data passes along the data lines out of the ALU the positive and zero flags are set according to the data. The carry flag is not affected. Thus in the first cycle of a conditional jump instruction the positive and zero flags may change from their state at the beginning of the instruction. If the flags were set by a non-accumulator instruction (a register instruction) their values may be corrupted during the first cycle of the next instruction. This is particularly important with consecutive conditional jumps.

For example:

```
DEC REG1     decrement register one
JIZ ZERO     jump if zero to ZERO
JIP POS      jump if positive to POS
```

The positive and zero status flags are set by the DEC instruction and tested by the JIZ instruction. If the test is unsuccessful (i.e. result is not zero) control passes to the JIP instruction and unless the contents of the accumulator are exactly thesame as the data which set the flags (i.e. contents of register one), the positive and zero flags will be corrupted by the JIZ instruction. Thus the JIP instruction may give an unexpected result.

The safe combinations of jumps are shown by an asterisk in the table and unsafe combinations by an exclamation mark.

		FIRST INSTRUCTION			
		JIZ JNZ	JIP	JRS	JCS JCN
SECOND INSTRUCTIONS	JIZ JNZ	!	!	!	!
	JIP	!	!	!	!
	JRS	*	*	*	*
	JCS JCN	*	*	*	*

!	Unsafe
*	Safe

Table - Combinations of Conditional Jumps

The positive and zero flags are dynamic whereas the carry flag is a flip-flop only altered under special conditions (see Note 2). The JRS instruction (jump if register S is not equal to seven) tests a logical AND of the bits of register seven and is thus unaffected by data on the data lines (unless S is being changed).

Note 2 - Setting the Carry Flag

The Carry Flag is a BCD carry flag and is affected by four instructions only. The instructions are BCD add (ADR), compare (CMP), single bit left shift accumulator LSA and single bit right shift accumulator RSA. Left and right shift instructions, respectively, set and clear the carry flag. ADR and CMP set the flag according to the result of their operation.

NOTE 3 - The Halt Instruction HLT

The Halt instruction is a simulator instruction only and is not implemented in the processor. It terminates a simulation of a program, causing the Simulator to print out the contents of the 48 processor registers. The effect of the HLT instruction op-code in the processor is undefined and the instruction should not be used (see note 4).

NOTE 4 - No Operation Instruction NOP

Three more instruction op-codes produce undefined results in the processor and should not be used. They have been given the mnemonic NOP, although the Assembler does not recognise it. In the Simulator a NOP instruction is skipped.

NOTE 5 - Register Addressing

The register instructions contain a four-bit code which identifies the register to be accessed. Now four bits can uniquely address only 16 registers and so a special convention is assumed by the processor. Two three bit registers S and T point to the row and plane respectively of the array of 48 registers to identify one particular register. S can take values in the range 0-7 (8 rows) leaving T with legal values 0-5 (6 planes). S can be incremented, decremented and loaded, T can only be loaded.

When the processor finds a register instruction which attempts to access register 12 it assumes indirect register mode and accesses the register pointed to by S and T. Register S is autodecremented after the indirect access, when a direct access of register 13 is attempted and autoincremented after the indirect access when a direct access of register 14 is attempted. The table below illustrates these points. Note that a direct access to register 15 is undefined.

Register "Directly Accessed"	Register Actually Accessed
0 - 11	0 - 11
12	register pointed to by ST
13	register pointed to by ST. S decremented.
14	register pointed to by ST. S incremented.
15	BEWARE! Undefined.

NOTE 6 - Loading Registers S and T

Registers S and T can be loaded with 3 bit short literals by the instructions LSS and LTS. Bits 1-3 of the instruction words are loaded into the appropriate register. The instruction SAT will store the accumulator bits 1-3 in register T but the instruction SST will store accumulator bits 4-6 in register T and bits 1-3 in register S. This point must be watched when loading T from the accumulator.

The instruction SAT can be used to "switch" the indirect register access from plane to plane. A whole row of registers can be accessed by changing T. When used with autoincrement or autodecrement indirect accessing, whole planes of registers can be manipulated with great ease.

5.5 DEFINITION OF INSTRUCTION SET OPERATION

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				
					Control	Address Bus		Data Bus	Program Counter
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB (1-3)		
LAL	0000 0100 **** ****	Load accumulator with long literal	A:=A	–	0 0 0 0	YER(4-6)	No Significance IR 1-3	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
			A:=P	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
ALL	0000 0101 **** ****	Logical AND, accumulator with long literal	A:=A	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
			A:=A <u>and</u> P	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
EOL	0000 1100 **** ****	EXCLUSIVE OR, accumulator with long literal	A:=A	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
			A:=A <u>exor</u> P	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
ORL	0000 1101 **** ****	Logical OR, accumulator with long literal	A:=A	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
			A:=A <u>or</u> P	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
ALA	0000 1110 **** ****	Add literal to accumulator	A:=A	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
			A:=A + P	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
CMP	0000 1111 **** ****	Compare accumulator with long literal	A:=A	–	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
			A + P sets U8	2	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

	INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
						Control	Address Bus		Data Bus	
	MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB (1-3)		
Instructions	LAS	1111 ****	Load accumulater with short literal	A(1-4):=P(1-4) A(5-8):=0	-	0 0 0 0	YER(4-6)	No significance IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	LSS	0010 1***	Load S with short literal	S:=P(1-3)	6	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	LTS	0011 1***	Load T with short literal	T:=P(1-3)	6	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
S and T Instructions	SST	0000 0011	Store accumulator in register ST	S:=A(1-3) T:=A(4-6)	6	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	SAT	0000 0001	Store accumulator in register T	T:=A(1-3)	6	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
					Control	Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB(1-3)		
LSA	0001 1100	Shift accumulator 1 bit left	A(2-8):=A(1-7) A(1):=0 U8:=1	2	0 0 0 0	YER(4-6)	No significance IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
RSA	0001 1101	Shift accumulator 1 bit right	A(1-7):=A(2-8) A(8):=0 U8:=0	2	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
LSN	0001 1110	Shift accumulator 4 bits left	A(5-8):=A(1-4) A(1-4):=0	-	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
RSN	0001 1111	Shift accumulator 4 bits right	A(1-4):=A(5-8) A(5-8):=0	-	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

	INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
						Control	Address Bus		Data Bus	
	MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB(1-3)		
Arithmetic and Logical Operations	LAR	1000 ****	Load accumulator from register	A:=RAM(addr)	5	0 0 0 0	YER(4-6)	No significance IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	SAR	1001 ****	Store accumulator in register	RAM(addr):=A	5	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	BAD	1010 ****	Add register to accumulator (binary)	A:=A+RAM(addr)	5	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	AND	1011 ****	Logical AND, register with accumulator	A:=A <u>and</u> RAM(addr)	5	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	EOR	1100 ****	EXCLUSIVE OR, register with accumulator	A:=A <u>exor</u> RAM(addr)	5	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	DEC	1101 ****	Decrement register by one	RAM(addr):=RAM(addr)-1	5	0 0 0 0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
	ADR	1110 ****	BCD add accumulator to register	RAM(addr):=RAM(addr)+A+U8 sets U4 and U8	2, 5	0 1 0 0	YER(4-6)		1111 1111 no meaning	-
				If U4=0 then RAM(addr)_{1-4}=RAM(addr)_{1-4}+10 If U8=0 then RAM(addr)_{5-8}=RAM(addr)_{5-8}+10	5	0 0 0 0	YER(4-6)		IR:=MEM(Q)	Q:=Q+1

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES							Program Counter
					Control				Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO	CDA	CQZ	CRA	ADB(4-6)	ADB(1-3)		
LAV	0000 1000	Load accumulator from register V	A:=RAM(12)	-	0	0	0	0	YER(4-6)	No significance IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
LAW	0000 1001	Load accumulator from register W	A:=RAM(13)	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
LAX	0000 1010	Load accumulator from register X	A:=RAM(14)	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
LAY	0000 1011	Load accumulator from register Y	A:=RAM(15)	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SAV	0001 1000	Store accumulator in register V	RAM(12):=A	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SAW	0001 1001	Store accumulator in register W	RAM(13):=A	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SAX	0001 1010	Store accumulator in register X	RAM(14):=A	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SAY	0001 1011	Store accumulator in register Y	RAM(15):=A	-	0	0	0	0	YER(4-6)		IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
					Control	Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB(1-3)		
SZX	0001 0010	Store accumulator in Z, module X	P:=A	-	0 1 0 0	YER(4-6)	IR(1-3)	P	-
			A:=A		1 1 1 0	X(4-6)	X(1-3) Z(9-11):=X(1-3)		-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SZY	0001 0011	Store accumulator in Z, module Y	P:=A	-	0 1 0 0	YER(4-6)	IR(1-3)	P	-
			A:=A		1 1 1 0	Y(4-6)	Y(1-3) Z(9-11):=Y(1-3)		-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SQX	0001 0110	Store accumulator in Q, module X	P:=A	-	0 1 0 0	YER(4-6)	IR(1-3)	P	-
			A:=A		1 1 0 0	X(4-6)	X(1-3) Q(9-11):=X(1-3)		-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SQY	0001 0111	Store accumulator in Q, module Y	P:=A	-	0 1 0 0	YER(4-6)	IR(1-3)	P	-
			A:=A		1 1 0 0	Y(4-6)	Y(1-3) Q(9-11):=Y(1-3)		-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
					Control	Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQ2 CRA	ADB(4-6)	ADB(1-3)		
LAM	0010 0***	Load accumulator from module (*** + 56)	A:=A	-	0 0 0 0	1 1 1	IR(1-3)	P:=AR or DATA	-
			A:=P	-	0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SAM	0011 0***	Store accumulator in module (*** + 56)	P:=A	-	0 1 0 0	1 1 1	IR(1-3)	P	-
			A:=A	-	1 0 0 0	1 1 1	IR(1-3)	AR:=P	-
			A:=A	-	0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
					Control	Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB (1-3)		
LIX	0000 0110	Load accumulator with (Z),	A:=A		0 0 1 0	X(4-6)	X(1-3)		-
		module X	A:=A		0 0 1 0	X(4-6)	X(1-3)		-
LIM		Load accumulator from peripheral	A:=P	-	0 1 0 0	YER(4-6)	IR(1-3)		-
		module X.	A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
LIY	0000 0111	Load accumulator with (Z),	A:=A		0 0 1 0	Y(4-6)	Y(1-3)		-
		module Y	A:=A	-	0 0 1 0	Y(4-6)	Y(1-3)		-
			A:=P		0 1 0 0	YER(4-6)	IR(1-3)		-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
SIX	0000 0010	Store accumulator in (Z),	P:=A		0 1 0 0	YER(4-6)	IR(1-3)	P	-
		module X	A:=A	-	1 0 0 0	X(4-6)	X(1-3)		-
SIM		Store accumulator in peripheral, module X.	A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
					Control	Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB(1-3)		
JMP	0100 0### #### ####	Unconditional Jump	A:=A		0 0 0 0	YER(4-6)	IR(1-3)	P:=MEM(Q)	Q:=Q+1
			A:=A	1	1 1 0 0	YER(4-6)	Q(9-11):=IR(1-3)	Q(1-8):=P	-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
JIZ	0100 1### #### ####	Successful Conditional Jumps	A:=A		0 0 0 0	YER(4-6)	IR(1-3)	P:=MEM(Q)	Q:=Q+1
JNZ	0101 0### #### ####	See Footnote	A:=A	1	1 1 0 0	YER(4-6)	Q(9-11):=IR(1-3)	Q(1-8):=P	-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
JIP	0101 1### #### ####	Non-satisfied Conditional Jumps	A:=A	1	0 0 0 0	YER(4-6)	IR(1-3)	P:=MEM(Q)	Q:=Q+1
JRS	0110 0### #### ####		A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
JCS	0110 1### #### ####	See Footnote							
JCN	0111 0### #### ####								

There are six conditional jumps:- jump if zero (JIZ), jump if not zero (JNZ), jump if Positive (JIP), jump if register S does not contain seven 'JRS), jump if carry set (JCS) and jump if carry not set (JCN).
If the jump condition is satisfied, the instruction believes as a normal unconditional jump (3 cycles).
If the condition is not satisfied the instruction takes only two cycles and does not change the sequence of the program.

INSTRUCTION			INTERNAL OPERATION	See Note	EXTERNAL DATA AND ADDRESS BUSES				Program Counter
					Control	Address Bus		Data Bus	
MNEMONIC	OP-CODE	FUNCTION			CIO CDA CQZ CRA	ADB(4-6)	ADB(1-3)		
GOS	0111 1*** **** ****	Go to subroutine (push stack)	A:=A	1	0 0 0 0	YER(4-6)	IR(1-3)	P:=MEM(Q)	Q:=Q+1
			A:=A		1 1 0 1	YER(4-6) * Z:=RB * RB:=RA * RA:=Q *Q(9-11):=IR(1-3)	IR(1-3)	Q(1-8):=P	-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1
RET	0000 0000	Return from subroutine (pull stack)	A:=A	1	0 1 0 1	YER(4-6)	IR(1-3)		-
			A:=A		0 0 0 0	YER(4-6)	IR(1-3)	IR:=MEM(Q) P:=MEM(Q)	Q:=Q+1

5.6 HARDWARE SYMBOLS

B	Instruction Register	
P	Data Buffer	
A	Accumulator	
YER	Y extension Register	
Q	Program counter and top stock register	)
RA	Second stack register	)
RB	Third stack register	) Subroutine stack
Z	Data address register	)
RAM	CPU registers	
V	CPU register 12	
W	CPU register 13	
X	CPU register 14	
Y	CUP register 15	
U4	BCD carry flag - lower digit	
U8	BCD carry flag - upper digit	
BN	Result Zero flag	
BP	Result positive flag	
MEM	Memory (external to the CPU chip)	

MEANING OF SYMBOLS USED IN THE DEFINITIONS

:=	"becomes"	This symbol means that the contents of the register/accumulator on the left-hand side becomes equal to the value of the expression on the right-hand side.
ARRAY [PTR]		Means the contents of the register/memory location pointed to by PTR. ARRAY is the name of the memory or register array. When ARRAY is a register name PTR will be a bit number or a bit range, in which case, the symbol will mean the contents of the appropriate bits. Thus RAM [addr] is the RAM register (in the processor) pointed to by addr. Where addr is calculated according to the formula:- if the argument (bits 1-4) of the instruction lie in the range 0-11 then addr equals the argument (i.e. direct register access). if the argument is 12 the access is indirect and addr is given by the contents of registers T and S. if the argument is 13 the access is again indirect via S and T, but S is decreased by one after the access. if the argument is 14 the access is indirect via S and T, and S is incremented after the access.
;	semicolon	Indicates that the two activities on either side of the semicolon occur simultaneously. When an activity is the name of a status flag this means that the flag is set by the contents of the data lines during this machine cycle, i.e. A:=A; U4; U8 means that the accumulator contents are cycled around the data lines and U4 (first BCD carry) and U8 (second BCD carry) are set according to the accumulator contents.
:	colon	The activity following the colon is dependent upon the status flag before the colon. If the flag is set the activity is performed.

When an instruction is fetched from ROM memory the Q register (program counter) points to the word to be accessed. The Q register is the top register of the stack in a 2K page. The pnogram counter is the Q register in the page containing the program. It is incremented after each fetch.

The fetched instruction is loaded into the instruction register B and, simultaneously, into the data buffer P. The P register is accessed when a data literal must be removed from the instruction word. During a jump instruction the wecond word of the instruction is fetched (using the Q register) into the P register. The complete address is assembled from part of the instruction (in the instruction register B) and the contents of the P register.

Thus the P register performs two functions - allowing the arithmetic and logic unit to access part of the instruction word, and providing a buffer for words fetched from the ROM to be put onto the address lines.

A similar technique of providing an image of a register, to be accessed by other parts of the processor circuits, is used with the Y register in its function as a module addressing register. A three bit register YER is loaded with an image of bits 4, 5 and 6 of the Y register. The contents of the YER register are then put onto the address bus lines ADB6X, ADB5X and ADB4X when a program address is needed. The YER register is used in preference to the Y register because of the increased speed of accessing.

The symbol MEM is used to represent the 2K page of ROM selected by the module address contained in the YER register. Thus the expressions

B:=MEM[Q]

P:=MEM[Q]   Q:=Q+1

mean that an instruction word if fetched (using the Q register) from the currently selected program ROM page and loaded into the instruction register and the data buffer register. The program counter (Q register) is then incremented.

Two static flags U4 and U8 are used to indicate four and eight bit BCD carries. During a BCD add instruction the flags are set or cleared, as appropriate, in the first cycle of the instruction. They/

They are used during the second cycle of the instructions to provide BCD correction.

The control signals CIO,CDA, CQZ and CRA are held on the four control bus lines. CIO controls input/output, CDA is the data or address control signal, CQZ specifies whether the Q register or the Z register is used during a ROM access and CRA controls the movement of the stack.

Software

Manual

SECTION INDEX

SECTION A

THE ASSEMBLER

INDEX OF CHAPTERS

CHAPTER ONE

FUNDAMENTAL COMPUTER OPERATION

Computers (large, mini- and micro-) consist basically, of a processor, memory and some input/output devices. The processor executes instructions of a program which are in sequence in memory. The instructions are a fixed length string (a word) of bits and can be regarded as just a collection of parallel inputs to a complex logic network, which is the processor. The instructions are fetched from memory by the processor (the last act a processor performs when executing an instruction is to fetch the next). The memory is a list of words each one of which can be individually accessed by specifying its address. The address is held in the program counter which is incremented after each instruction fetch. The processor can fetch data from and store data in any location of the memory. It can also change the program counter, effectively "jumping" about the program by fetching the next instruction from the newly-addressed location.

/...2

Example 1.1

Address	Instruction Mnemonic	Address of Operand	Meaning
0100	LD	102	Loads the contents of location 102 = 6
0101	JMP	103	Jumps over location 102 to 103
0102	6		Data
0103	ST	109	Stores 6 in location 109

The above program, in a simple fictitious language, loads the contents of one location into a register and then stores them in another location. If the user decided that he should subtract, say 5 from the data before storing it in the second location the program would become:-

Address	Instruction Mnemonic	Address of Operand	Meaning
0100	LD	103	Load 6
0101	SUB	104	Subtract 5
0102	JMP	105	Jump over data
0103	6		Data
0104	5		Data to be subtracted
0105	ST	?	Store in location?

Every address after the inserted instructions must be changed and some addresses before must be changed to allow the program to execute as required. Note that data and instructions can both be stored in the same memory, the JMP instruction in the above program is required so that the data in location 103 (i.e. 6) is not fetched by the processor and interpreted as an instruction. In general the data and program as physically separated in separate parts of memory or avoid mixing, this generally tends to make software more reliable.

1.1 Justification of An Assembly Language

The ALPS processor responds to, and executes, instructions which are stored in a memory as a string (a word) of binary 0's and 1's. While these strings have meaning to the processor they are confusing for the user and some form of symbolic representation (Mnemonic) is required. The assembly language exchanges the machine code (i.e. 0's and 1's) for instruction mnemonics, which are short meaningful strings of characters. The user can select mnemonics, or names, which characterize the nature of the instruction and can write fairly readable programs using them. To obtain the 0's and 1's that the processor needs, the user can convert each mnemonic of his program by consulting a table, to the appropriate machine code - a laborious procedure - or he can use a computer program, the assembler, to convert them for him.

At this stage the user program consists of mnemonics and numbers, which are converted by the assembler to machine code instructions and data. Consider the situation of a user with a program consisting of mnemonics and numbers, who wishes to insert some more instructions in the middle of his program. If the numbers are addresses (say - destinations of jump instructions) the entire program may have to be modified to account for the change of addresses caused by the introduction of new instructions. See example 1.1.

This is obviously a very time-consuming procedure fraught with possible errors and to sidestep it we let the assembler check the addresses for us. Instead of using absolute addresses we use symbols, which can be meaningful names that the assembler can recognise and assign values to. This, as well as reducing errors and simplifying modifications, makes the program more readable as shown in example 1.2. The symbols can have values assigned to them by the assembler as in STOCK and REQD, or they can have their values assigned by the user in a statement of the form shown in example 1.3. Thus constants can be assigned/

Example 1.2

Address Label	Instruction Mnmmonic	Name of Operand	Meaning
0100	LD	STOCK	Loads STOCK = 6
0101	JMP	OVER	Jumps over data
0102 STOCK	6		Data
0103 OVER	ST	SPARE	

and the modified program

Address Label	Instruction Mnemonic	Name of Operand	Meaning
0100	LD	STOCK	Load STOCK
0101	SUB	REQD	Subtract REQD
0102	JMP	OVER	
0103 STOCK	6		Data
0104 REQD	5		Data
0105	ST	SPARE	

Example 1.3 User Assigned Symbols

```
THREE  :  EQUALS 3
LABEL1 :  EQUALS 6
EXIT   :  EQUALS LABEL5
```

assigned symbols without necessarily being actual labels.

The assembler can be made to read the label symbol, the instruction mnemonic, the operand symbol and then ignore everything else in the input line. Comments can then be inserted after the operand symbol to illustrate the operation of the program without affecting the interpretation of the assembler.

A further advantage of using an assembler lies in the choice of number systems available. At machine-code level the numbers must be binary (0's or 1's) but at assembly language level we can use binary, octal, decimal, binary coded decimal or hexadecimal systems. The choice of default number system will depend upon the word size.

CHAPTER TWO

INTRODUCTION TO THE ALPS ASSEMBLY LANGUAGE

Source Program Format

CHAPTER TWO

INTRODUCTION TO THE ALPS ASSEMBLY LANGUAGE

Source Program Format

2.1 Statement Format

A statement can consist of up to four fields which are identified by order of appearance and special terminating characters. The general format of a ALPS assembly language statement is:-

label: operator operand comment

The label and comment fields are optional. The operand field is dependant upon the type of the operator. A statement must contain at least an operator and may contain any of the other types.

The assembler interprets and processes these statements one by one, generating one binary (machine-code) instruction and sometimes a data word.

2.2 Label Field

A label is a user-defined symbol and is assigned a value by the assembler. The value assigned will be the address of the current location, i.e. the address of the instruction as in Example 1.2, or will be obtained from the operand field in the case of a user-assignment as in Example 1.3.

A label is a symbolic means of referring to a specific location within a program. If present a label always occurs first in a statement and must be terminated by a colon. There can be only one label per line. The first six characters of a label are significant and can be preceded by any number of spaces.

Trailing/

Trailing spaces between the seventh label character and the colon are ignored. Any characters except colon can be used in a label but alphanumeric labels are recommended for clarity.

A symbol used as a label cannot be redefined in a user program.

2.3 Operator Field

The operator field follows the label field (if any) in a statement, leading spaces and tabs are ignored. The operator consists of up to six characters, starting at the first non-space character after the label field colon (or start of line) and terminated by a space. It can be an instruction mnemonic or an assembler directive. When it is an instruction mnemonic, it speci"ies the instruction to be generated and the action to be performed on the operand which follows (if any).

2.4 Operand Field

This field is the last field used by the assembler and follows the operator field, intervening spaces being ignored. It can be up to eight characters long and is terminated by a space, except when the operand is an ASCII string. Such a string is enclosed in single quotes and can extend up to the end of the statement line. The operand field can be optionally followed by a comment, separated by one or more spaces. The operand is that part of the statement that is manipulated by the operator field. Operands can take one of the following forms:-

(a) decimal number

(b) hexadecimal number

(c) label (of up to six characters in length)

(d) * (current location))

(e) * ± decimal number) see Section 3.12

(f) * ± hexadecimal number)

(g) @)

(h) @ +) indirect register addressing - see Section 3.9

(i) @ -)

(j) , (comma - null operator) - see Section 2.5

2.5/

2.5 Comment Field

This field is optional and may contain any ASCII characters. It is ignored by the Assembler and can be used to document the program. The comment can be the only field on the line or can follow the operand field. If the first character on the line is a number sign # the entire line is treated as a comment. If the operand field contains a comma, the Assembler treats this as a null operator replacing it with spaces. It then treats all further characters as a comment.

2.6 Format Control

Horizontal formating of the source program is controlled by space or tab (which is transliterated to space). These characters have no effect upon the assembly except when they serve as field delimiting characters. Vertical formating is controlled by the form-feed character. Two assembler directives are provided for vertical formating of the assembly listing which is horizontally formated and paged by the assembler.

See Appendix C for examples of listings.

CHAPTER THREE

PROGRAMMING IN THE ALPS ASSEMBLY LANGUAGE

Index

CHAPTER THREE

PROGRAMMING IN THE ALPS ASSEMBLY LANGUAGE

This chapter describes the components of legal assembly language statements.

3.1 Character Set

A subset of the ASCII character set (see Appendix A) is used in source program statements and includes:-

(a) the upper-case alphabet A-Z

(b) the digits 0-9

(c) the following characters:

Character	Name	Function
Carriage Return		formating character statement terminators
Line Feed		formating character statement terminators
Form Feed		formating character statement terminators
:	colon	label terminator
#	number sign	comment field originator
,	comma	null operand field indicator
space		field terminator
tab		field terminator
/	slash	precedes a hexadecimal constant
*	asterisk	current location counter
+	plus sign	arithmetic addition operator, or post autoincrement (indirect register mode)
-	minus sign	arithmetic subtraction operator, or post autodecrement (indirect register mode)
@	at sign	indirect register mode
'	single quote	ASCII string delimiters
.	period	no present function
$	dollar sign	no present function;reserved for system use.

3.2 Separating and Delimiting Characters

As stated above, comma, semicolon, colon, single-quotes and space are field delimiters and should be used with care in any other conditions.

3.3 Illegal Characters

There are currently no illegal characters (except those used out of context). However, it is strongly advised that only uppercase teletype compatible ASCII characters should be used when writing programs.

3.4 Operator Characters

Two arithmetic operators + and - are used in the assembly language in two contexts:-

(a) used with the current location counter symbol * (see Section 3.13)

(b) used with the indirect register access symbol @ (see Section 3.9)

3.5 Permanent Symbols

These are the instruction mnemonics and assembler directives and are a permanent part of the Assembler. They need not be defined before assembly. Their definitions are relevant only to the operator field, the same symbols can be defined by the user as label symbols although this could make the resulting program confusing to read .

3.6 User Defined Symbols

User defined symbols are those used as labels or those defined by direct assignment. They are added to the symbol table as they are encountered during the first pass of the assembly.

3.7 Direct Assignment

A direct assignment statement associates a symbol with a value. When a direct assignment statement is encountered the symbol is entered into the symbol table and the specified value associated to it.

A direct assignment statement takes the form

SYMBOL: EQU expression

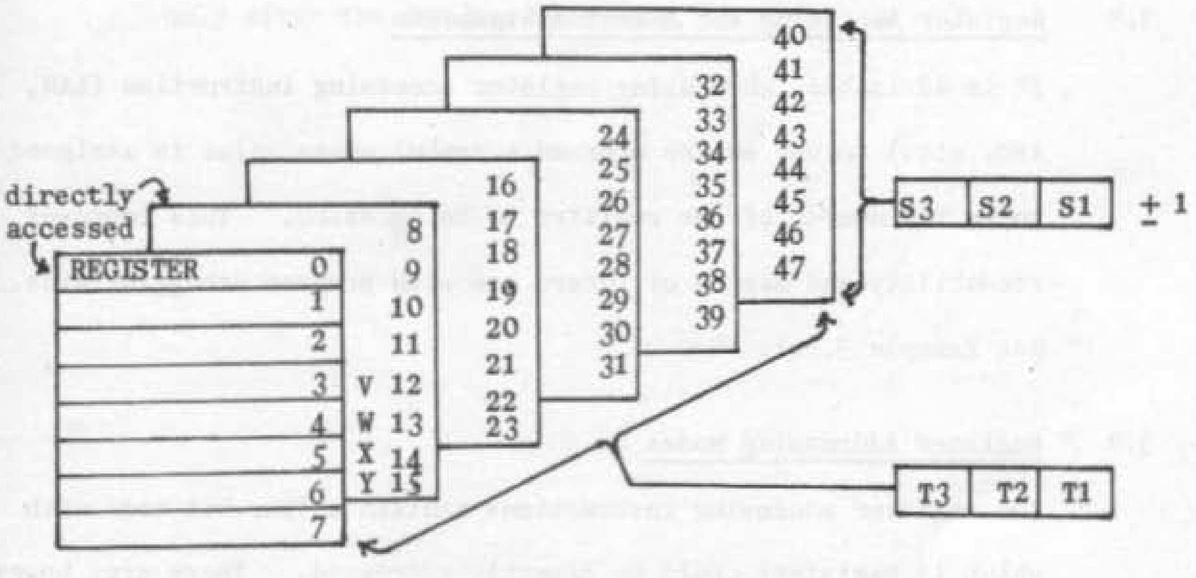
where/

```
    REG 1:  EQU 1

    REG 2:  EQU 2
            '
            '
            '
            '
            LAR  REG 1

            AND  REG 2
```

The above is more readily understood than:-

```
            LAR 1

            AND 2
```

EXAMPLE 3.8.1 Register Accessing and Direct Assignments



EXAMPLE 3.9.1 Register Arrangements

where SYMBOL is the symbol to be assigned (delimited by a colon), EQU is the assembler directive denoting direct assignment (see Section 4.2 and 4.3) and expression is one of operand expressions

(a) decimal number

(b) hexadecimal number

(c) label

(d) * current location

(e) * ± decimal number

(f) * ± hexadecimal number

<u>Note</u>: that an assignment does not take up any program space, it merely associates a value with a more convenient or descriptive name. Only one symbol can be assigned per direct assignment statement and it cannot be redefined, nor can a direct assignment reassign a previously declared label.

No forward referencing is allowed with direct assignments.

3.8 <u>Register Accessing and Direct Assignments</u>

It is advisable, when using register accessing instruction (LAR, AND, etc.) to use as the operand a symbol whose value is assigned to be the number of the register to be accessed. This improves readability and may be of future use with program debugging aids. See Example 3.8.1.

3.9 <u>Register Addressing Modes</u>

The register accessing instructions contain a four bit code with which 15 registers could be directly addressed. There are, however, 48 registers on the processor chip and to allow access to all of them two three-bit registers S and T are used. The 48 registers are arranged as an array of 8 rows and 6 planes (-the eight bits of each register forming the columns). See Example 3.9.1. Register T addresses the plane of the array and register S the row.

Both/

Register "directly accessed"	Register actually accessed	Action on S
0–11	0–11	none
12	RAM [ST]	none
13	RAM [ST]	decremented
14	RAM [ST]	incremented
15 Beward!	Undefined	undefined

where RAM [ST] means the register pointed to by S and T.

TABLE 3.9.2 Summary of Register Addressing

Note: When addressing modes 13 and 14 are used the RAM location accessed is the one that's address is the value of the S,T register before the instruction execution was started. The S register contents are modified, decremented or incremented, only after the RAM has been used.

Both registers can be loaded from the accumulator or with three bit literals. Register S can be autoincremented, autodecremented or left unchanged according to the instruction addressing mode. To allow for these three indirect modes of register access,three of the direct modes are given special meaning.

If an attempt is made to directly access register 12, 13 or 14 the processor traps it and interprets the instructions indirect, autodecrement indirect or autoincrement indirect register access. Table 3.9.2 summarises the above. A direct access of register 15 gives undefined results. Registers 12, 13 and 14 are not affected by a direct access on them.

Indirect accessing causes the register pointed to by S T to be accessed, autodecrement indirect accessing reduces S by one after the access and autoincrement accessing increases S by one after the access. Thus all 8 registers of a plane can be indirectly accessed by successively autoincrementing or autodecrementing S.

To clarify, on the assembler listing, the use of indirect register addressing a special symbol used. When the assembler finds an "at sign" (@), after a register access mnemonic, it assumes indirect addressing. It then searchs for an operator symbol (+ or -) and assembles them as:-

Symbol	Meaning	Instruction Argument
@	indirect addressing	12
@ -	autodecrement indirect	13
@ +	autoincrement indirect	14

Any other use of @ or arrangement of @ and + or - is illegal.

To allow registers 12, 13, 14 and 15 to be directly accessed they are given names (V, W, X and Y) and eight instructions are provided for loading and storing to and from accumulator and register.

See Chapter Four - Instruction Set.

3.10/

3.10 Macro and System Routine Calls

Two characters, the period (.) and the dollar sign ($), are reserved for future expansion of the assembler to include macro and system routine calls. The macro (or routine name) preceded by a period (or a dollar sign) would be treated as an assembler directive, to cause the assembler to insert suitable code.

3.11 The Number Systems

The default number system of the assembly language is decimal. Operands can also be given as hexadecimal numbers. Hexadecimal numbers are preceded by a slash and contain the characters 0-9, A-F, where:-

A	=	10
B	=	11
C	=	12
D	=	13
E	=	14
F	=	15

Two hexadecimal numbers can be packed into one eight bit word. The largest hexadecimal number that can be held is FF (= 255). See Example 3.11.1.

The processor instruction set includes an instruction for adding together Binary Coded Decimal (BCD) numbers. BCD numbers are a subset of the hexadecimal system. One decimal digit is stored in four bits and six of the possible sixteen arrangements of the four bits are unused. Two BCD digits are stored in one word. The conversion of a number stored in BCD for input or output is very simple and for this reason the redundancy of the number system is acceptable.

3.12 The Program Counter and The Data Address Register

The program counter Q and the Data Address register Z are eleven bits long and thus can be used to address 2048 (2K) individual 8 bit locations of memory. During execution of an instruction the program/

		Decimal	Internal Representation
Decimal	104	104	0110 1000
Hexadecimal	/1A	26	0001 1010

EXAMPLE 3.11.1 Number Systems

program counter points to the next instruction to be executed. At the end of an instruction the next instruction is fetched from memory into the instruction register IR, and then the program counter is incremented (increased by one). The program counter can be modified under program control to cause a "jump" to another part of the program.

If the user wishes to store data in or load data from memory, the program must load the data address register Z with the address of the location to be accessed, and then access the location with a load or a store instruction (see Chapter Four - Instruction Set).

3.13 Jump Instructions

A program "jumps" when the processor is caused to execute, not the next sequential instruction, but another one elsewhere in the program. This is achieved by changing the contents of the program counters so that the next instruction to be executed is fetched into the processor from the destination of the jump. The program counter can be directly modified by the processor (as in a jump instruction) or indirectly by an instruction which stores a value in the counter. In the former case the destination address is determined at assembly time, whereas in the latter it is computed at runtime. The latter technique (hand-loading the program counter) can be used to provide an indirect jump via a list of routine names. Extreme care must be taken, when modifying the program counter in such a fashion, to ensure that the processor jumps to the start of an instruction (and not to the second word of a two word instruction or into data). The source program can be written entirely using labels as destinations for jumps or by using relative jumps. In relative jumps the address is the program counter plus or minus an offset, although it is assembled to an absolute 11 bit address. The operand of the instruction takes one of the three forms:-

(i)/

(i) * current location

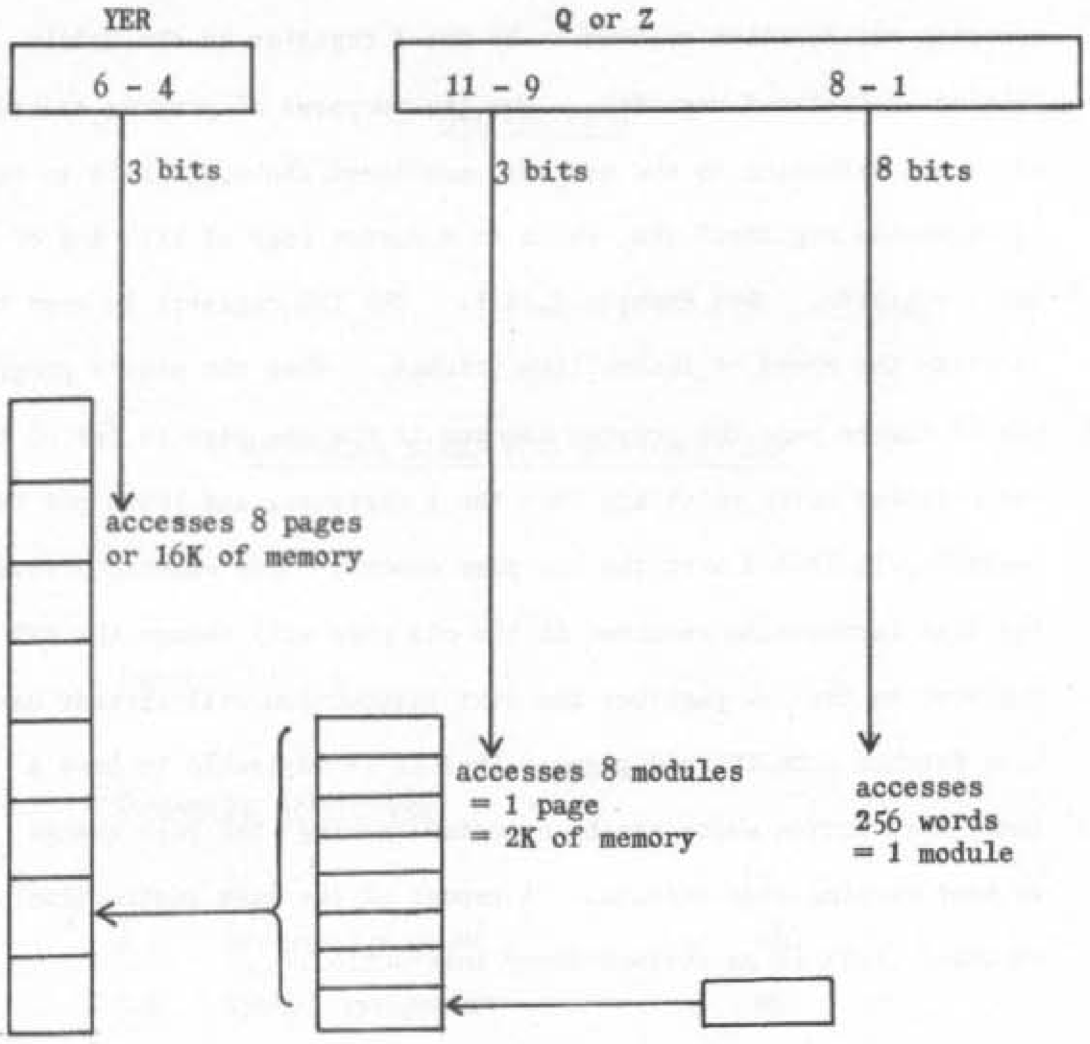
(ii) * ± decimal number

(iii) * ± hexadecimal number.

When an asterisk is encountered alone the instruction argument is the contents of the program counter. If a plus or a minus is found the following symbols are scanned to assemble the offset (which can be given in decimal or hexadecimal). The offset is then added to or subtracted from the program counter to give the instruction argument. These forms of operand can be used with most instructions, not just the jump instructions. It is important that the consideration of the destination address, given in the case of "hand-loading" the program counters be given when using program counter offsets.

3.14 Activities Outside of a 2K Area

To overcome the limitations imposed by a 2K range of addresses the load, store and jump instructions all assume a 3 bit extension to the Q or Z registers. This makes them effectively 14 bits (although in the case of the Q register - program counter - only the bottom 11 bits are actively incremented) allowing 16K of memory to be accessed. However with this simple system of one extension register, when a program is running in one 2K "page" it can only access locations in that 2K page. To access locations in other pages it must jump into those pages by changing the extension. This is crude and wasteful of space, so the processor is given a choice of two "module" selecting registers X and Y. (With an eight bit word 256 locations or one module can be uniquely addressed.) Each page of memory has its own Q and Z registers. The processor assumes that the program lies in the page addressed by the contents of the Y register. Data can be fetched from the program page by using an instruction to load the accumulator with the contents of the location pointed to by the Z register, on the module pointed to by the Y register. Data can be stored in or fetched from any page by using an instruction which accesses/



YER
6 - 4
Q or Z
11 - 9
8 - 1
3 bits
3 bits
8 bits
accesses 8 pages or 16K of memory
accesses 8 modules = 1 page = 2K of memory
accesses 256 words = 1 module

EXAMPLE 3.14.1 Addressing Scheme

To jump to a location in another page: the page number is held in bits 4-6 of N and the module number in bits 103. M is the address within a module.

LAL	N	load N into accumulator
SAX		store it in register X
LAL	M	load M into accumulator
SQX		set up Q 1-8 = M and Q 9-11 = X 1-3
LAX		reloads N into accumulator
SAY		set up Y (and YER) - jumps to new page
SAY		dummy instruction (see text)

EXAMPLE 3.14.2 Changing Pages

accesses the location pointed to by the Z register on the module pointed to by the X register. For the purposes of program counting the 3 bit extension to the address, mentioned above, is held in the "Y extension register" YER, which is a direct copy of bits 4-6 of the Y register. See Example 3.14.1. The YER register is used to increase the speed of instruction fetches. When the user's program has to change page the program counter in the new page is set up to the required entry point and then the Y register, and hence the YER register, is loaded with the new page number. See example 3.14.2. The last instruction executed in the old page will change the YER register to the new page but the next instruction will already have been fetched from the old page. Thus it is advisable to have a dummy instruction which can be executed "during" the page change without causing side effects. A repeat of the last instruction executed (SAY) is an obvious dummy instruction.

CHAPTER FOUR

ASSEMBLER DIRECTIVES AND MNEMONICS

Index

CHAPTER FOUR

ASSEMBLER DIRECTIVES AND MNEMONICS

This chapter describes the Assembler Directives and the Instruction Mnemonics. Directives are statements which cause the assembler to perform certain processing operations. They control program location, symbol assignment, data storage, assembly listings and assembly termination or interruption. Directives are held in the operator field. The label assignment directives must be preceded by a label, the data storage directives may be, and all others must not be.

Assembler Directives

4.1 Program Location - ORG

The directive ORG instructs the assembler that the program counter is to have a new value. This value is given in the operand field in decimal or hexadecimal form. The format of the statement is:

ORG   origin address   (comment)

where the comment field is optional. There may be any number of spaces before ORG, but at least one between ORG and the address. There may not be a label. If the first statement of a program is not an ORG directive the assembler assumes an origin of zero. See Example 4.1.1 and Appendix C.

4.2 Symbol Assignment - EQU

The direct assignment directive EQU is used to assign a value to a symbol. The symbol to be assigned must be in the label field (terminated by a colon) and the value, a decimal number, hexadecimal number/

	STATEMENT	PROGRAM COUNTER
	ORG 506	506
	ORG /AFF	2815
ENTRY:	ORG 3	illegal syntax
	ORG TYPE	illegal syntax

EXAMPLE 4.1.1 Use of the ORG Directive

number or a previously defined label, must be in the operand field. The format of the statement is:

```
SYMBOL:    EQU      decimal number         (comment)
                    hexadecimal number     (comment)
                    pre-defined label      (comment)
```

As always the comment field is optional.

The symbol is assigned a value equal to the bottom eight bits of the operator; that is, the Assembler extracts the intra-module part of the address (the operator).

See Example 4.2.1 and Appendix C.

4.3 Symbol Assignment - MODUL

The direct assignment directive MODUL assigns a value to a symbol in the same manner as EQU. The value assigned is a six bit number equal to bits 9-14 of the operand; that is, the MODUL directive causes the Assembler to extract the module number from the address (i.e. the operator). The statement format is:

```
SYMBOL:    MODUL      decimal number         (comment)
                      hexadecimal number     (comment)
                      pre-defined label      (comment)
```

See Example 4.3.1 and Appendix C.

4.4 Data Storage - DC

The Define Constant directive DC causes the Assembler to load the value of the operand (in the range 0 to 255) into the current location. The operand can be:

(i) Decimal number

(ii) Hexadecimal number

(iii) A previously defined label.

The format of the statement is

```
(symbol:)      DC      decimal number         (comment)
                       hexadecimal number     (comment)
                       label
```

See/

STATEMENT				SYMBOL	VALUE	OPERAND	VALUE
THREE:	EQU	3	decimal	THREE	3	-	3
HAY:	EQU	/A	hexadecimal	HAY	10	-	10
EXIT:	EQU	OUT	label	EXIT	40	OUT	40
MATHS:	EQU	ADD	label 255	MATHS	150	ADD	406
	EQU	START		illegal syntax			
BEGIN:	EQU	HEAD	illegal if HEAD is undefined				
ADD:	EQU	CALC	illegal if ADD is already defined				
TWO:	EQU	TWO	illegal because TWO is already defined.				

EXAMPLE 4.2.1 Use of the EQU Directive

STATEMENT			SYMBOL	VALUE	OPERAND	VALUE
THREE:	MODUL	3	THREE	0	-	3
LABEL:	MODUL	/AFF	LABEL	10	-	2815

EXAMPLE 4.3.1 Use of the MODUL Directive

PROGRAM COUNTER		STATEMENT	CURRENT LOCATION VALUE	SYMBOL	VALUE
107	TWELVE:	DC 12	12	TWELVE	107
		DC 10	10	-	-
2577	DATA:	DC /4C	76	DATA	2577
		DC /AFF	illegal operand - greater than 255		
		DC -4	illegal operand - less than zero		
		DC TWELVE	107	-	-
		DC 'STRING'	illegal operand		

EXAMPLE 4.4.1 Use of the DC Directive

See Example 4.4.1 and Appendix C. Only one location can be loaded per directive. The label and comment fields are optional.

4.5 Data Storage - ASCII

The ASCII directive informs the Assembler that the operand is an ASCII string enclosed in single-quotes, which is to be loaded one character per location in consecutive locations. The ASCII string in single-quotes, can be between 1 and 16 characters long. The statement format is:

```
(symbol:)    ASCII      'ascii string'      (comment)
```

See Example 4.5.1 and Appendix C.

All ASCII characters except carriage return, line feed and single quotes can be used in the ASCII string. DC directives must be used to load the internal representation of these three characters into memory. The internal representation used is the seven bit (bit 8 equals zero) standard code. See Appendix A.

4.6 Assembly Listing Directive - SPACE

The SPACE directive may have neither label nor operand (although it may have a comment if the comma or null operator device is used, see Chapter 2 Section 2.5). It causes the Assembler to insert one blank line in the output listing. See Appendix C. The statement format is:

```
SPACE            ( ,   comment )
```

4.7 Assembly Listing Directive - EJECT

Like the SPACE directive EJECT may have neither label nor operand but it can have a comment. The assembler sends a form-feed to the line-printer, to generate a page throw and to continue the listing at the top of a new page.

The statement format is:

```
EJECT            ( ,   comment )
```

See Appendix C.

4.8/

PROGRAM COUNTER	STATEMENT	LOCATIONS	CONTENTS	CHARACTER	SYMBOL	VALUE
3011	ASCII 'THE CAT '	3011	84	T	-	-
		3012	72	H	-	-
		3013	69	E	-	-
		3014	32		-	-
		3015	67	C	-	-
		3016	65	A	-	-
		3017	84	T	-	-
		3018	32		-	-
3019 STRINGZ:	ASCII 'SAT'	3019	83	S	STRINGZ	3019
		3020	65	A	-	-
		3021	84	T	-	-
	ASCII 12	illegal operand				
	ASCII	no effect				

EXAMPLE 4.5.1 Use of the ASCII Directive

4.8 Assembly Interruption - EOT

The EOT directive warns the Assembler that the source program continues on a new tape. It is used in long source programs to divide the paper tapes into manageable lengths. When an EOT is found the Assembler warns the operator via a message on the teleprinter, and then waits for him to load the next tape. When the tape is ready the operator must type CO to continue. It is advisable to include an ORG directive at the beginning of each tape in case a tape is loaded out of sequence.

The directive may have a comment (using the comma device) but neither label nor operand.

Statement format:

```
EOT          ( ,   comment )
```

See Appendix C.

4.9 Assembly Termination - END

An END statement must be the last statement of the source program. The directive cannot have a label or an operand (it can have a comment). The END statement terminates pass one of the assembly and initiates pass two, statements after the END being ignored. If an END statement is not found before the end of the file the Assembler prints a warning message on the lineprinter and then inserts an effective END in preparation for the second pass. See Appendix C.

4.10 Instruction Set Mnemonics

See the Hardware Manual for an explanation of the hardware concepts and capabilities of the machine. The instruction set mnemonics are listed below in family groups according to function. They can all have labels but not all can have operands. Whether a family can have operands is noted with the family description.

Accumulator/

Accumulator and Registers (operand)

LAL	load accumulator with a long (8 bit) literal
LAS	load accumulator with a short (4 bit) literal
LAR	load accumulator from register. See Chapter Three
SAR	store accumulator in register. Section 3.9

Registers S and T

LSS	load S with a short (3 bit) literal	(operand)
LTS	load T with a short (3 bit) literal	(operand)
SST	store accumulator in registers S and T	(no operand)
SAT	store accumulator in register T	(no operand)

Accumulator and Address Control Registers (no operand)

LAV	load accumulator from register V
LAW	load accumulator from register W
LAX	load accumulator from register X
LAY	load accumulator from register Y
SAV	store accumulator in register V
SAW	store accumulator in register W
SAX	store accumulator in register X
SAY	store accumulator in register Y
SZX	store accumulator in Z on module X
SZY	store accumulator in Z on module Y
SQX	store accumulator in Q on module X
SQY	store accumulator in Q on module Y

Logical Operations (operand)

ALL	logical AND, accumulator with long literal
AND	logical AND, accumulator with register
ORL	logical OR, accumulator with long literal
EOL	exclusive OR, accumulator with long literal
EOR	exclusive OR, accumulator with register

Arithmetic/

Arithmetic Operations (operand)

BAD	binary add register to accumulator
DEC	decrement register by one
ADR	BCD add acc mulator to register
ALA	add long literal to accumulator
CMP	compare accumulator with literal

Shift Operation (no operand)

LSA	shift accumulator 1 bit left
RSA	shift accumulator 1 bit right
LSN	shift accumulator 4 bits left
RSN	shift acc mulator 4 bits right

Input/Output Direct (operand)

LAM	load accumulator from module
SAM	store accumulator in module

Input/Output Indirect (no operand)

LIX	load accumulator with (Z) on module X
LIY	load accumulator with (Z) on module Y
SIX	store accumulator in (Z) on module X

Jumps within a Page (operand)

JMP	unconditional jump
JIZ	jump if zero
JNZ	jump if not zero
JIP	jump if positi e
JRS	jump if register S $\neq 7$
JCS	jump if carry set
JCN	jump if carry not set

Subroutine Instructions

GOS	go to subroutine (operand)
RET	return from subroutine (no-operand)

In addition to the above mnemonics the Assembler also recognises HLT which halts the Simulator Program but has no effect in the processor.

CHAPTER FIVE

CROSS ASSEMBLER - OPERATING PROCEDURE

The Cross-Assembler

CHAPTER FIVE

CROSS ASSEMBLER - OPERATING PROCEDURE

The Cross-Assembler

The CROSS-ASSEMBLER is designed to simplify the creation of programs for the computer kit. It converts the user's program, written in symbolic form, into the operation codes that the processor can execute. Thus it is possible to define labels, constants and instructions with easily identifiable names that are meaningful to the user. The Cross-Assembler is written in ANSI FORTRAN IV and requires about 18K of core memory for operation (when run on a PDP 11). It can be modified to run on computers with a smaller core memory, however, the standard version has been designed to run on the PDP11 and Sigma computers.

5.1 General Description of the Cross-Assembler

The Cross-Assembler operates in two passes. On the first pass it produces a fixed-format intermediary copy of the free-format source file and collects information about the symbols and the individual statements. The source program can be prepared on punched cards or paper-tape. In the second pass the instruction mnemonics are decoded and the binary equivalent of the statement constructed using information stored in the symbol tables.

The size of the source program that can be handled by the Cross-Assembler is limited by the size of the tables. Without modification the Cross-Assembler can assemble a program that:

(a) contains fewer than 500 input statements including assembler directives

(b)/

(b) has no more than 100 defined program labels

(c) shows fewer than 500 label cross references.

In addition the source program must not exceed 4K (4096) words of ALPS memory. The Cross-Assembler is written in Fortran in such a way as to facilitate changes to the specification, with regard to the maxipum number of statements or labels.

5.2 Initial Dialogue

The Cross-Assembler announces itself and asks the operator questions about listings, paper tape punching, simulator files and symbol tables. The answers to the questions are usually YES or NO or a number. When the correct response is neither YES, NO nor a number the CROSS-ASSEMBLER will prompt the operator. The dialogue and some answers are shown in Appendix B.

5.3 Listings and Error Messages

Some errors are detected during pass-one and appropriate messages are output by the Cross-Assembler. The messages are self-explanatory and are inserted after the line in error in the pass one listing. If the line has not been listed the error routine will list the line before outputting error messages. At the end of pass one the total number of errors detected is printed on the teletype and the line-printer. A warning message will be printed on the lineprinter if an END is not found before the end of the source program file, it is however only a warning and is not counted as an error. The Cross-Assembler always prints a pass-two listing and the errors detected in pass two are printed after the line in error. At the end of the assembly the total number of errors detected in the second pass is printed on the teletype and the line printer. Examples of listings and the effect of Assembler directives are given in Appendix C.

5.4/

5.4 Simulator File

The Cross-Assembler can prepare a file of the output of the assembly, as an image of the program in 4K of memory. The operator can decide not to create the file and hence sa e space on the backup storage de ice.

The creation of the simulator file is terminated as soon as an error is detected. By this means, an old correct copy of the simulator file is not destroyed by a new incorrect one.

The file is used by a Simulation program to simulate the execution of the program on a ALPS system. The Simulation program is written in ANSI FORTRAN IV and Digital Equipment's MACRO-11 for running on a PDP 11 equipped with a line printer.

5.5 Paper Tape Punching - Loader Tape

Optionally the Cross-Assembler can punch a paper which contains, in one of two formats, information describing the user program in core. The two formats are:

(i) character

(ii) binary.

The character format is used when the paper tape may have to be modified or patched using just a teletype. The contents of each location (each location specified in the source program) are printed as two modified hexadecimal characters (see Appendix D) using standard ASCII code. When the bottom four bits of the modified hexadecimal character are removed, they produce the decimal that the character represented. The binary format is used when a program is complete and unlikely to be changed. It has the advantage of producing tapes about one half as long as the character format. In binary format each character represents one eight bit word (1 track per bit). A count has to be given at the beginning of the tape of the total number of locations represented since there/

there is no way of encoding an end of tape signal. The first character of the tape is the format description (i.e. B for Binary, C for Character) and the last few characters are the checksum. The format of both tapes is shown in figure 5.4.1.

The paper tape is prepared for loading the program into memory using the absolute loader of the system programs. If an error occurs during assembly the punching of the tape is discontinued.

APPENDICES TO SECTION A

APPENDIX A

ASCII Character Set

Character	7 bit Octal	Decimal	Hexadecimal	Character	7 bit Octal	Decimal	Hexadecimal
Space	040	32	20	A	101	65	41
!	041	33	21	B	102	66	42
"	042	34	22	C	103	67	43
#	043	35	23	D	104	68	44
$	044	36	24	E	105	69	45
%	045	37	25	F	106	70	46
&	046	38	26	G	107	71	47
'	047	39	27	H	110	72	48
(	050	40	28	I	111	73	49
)	051	41	29	J	112	74	4A
*	052	42	2A	K	113	75	4B
+	053	43	2B	L	114	76	4C
,	054	44	2C	M	115	77	4D
-	055	45	2D	N	116	78	4E
.	056	46	2E	O	117	79	4F
/	057	47	2F	P	120	80	50
0	060	48	30	Q	121	81	51
1	061	49	31	R	122	82	52
2	062	50	32	S	123	83	53
3	063	51	33	T	124	84	54
4	064	52	34	U	125	85	55
5	065	53	35	V	126	86	56
6	066	54	36	W	127	87	57
7	067	55	37	X	130	88	58
8	070	56	38	Y	131	85	59
9	071	57	39	Z	132	90	5A
:	072	58	3A	[	133	91	5B
;	073	59	3B	\	134	92	5C
	074	60	3C	]	135	93	5D
=	075	61	3D	↑	136	94	5E
	076	62	3E	←	137	95	5F
?	077	63	3F	Carriage Return	015	13	D
@	100	64	40	Form Feed	014	12	C
				Line Feed	012	10	A

APPENDIX B

Cross-Assembler Dialogue

```
GIM LTD ALPS CROSS-ASSEMBLER V001C 13-DEC-71

DO YOU WANT THE OUTPUT LISTING ON THE LP OR TTY?

TTY

DO YOU WANT A PASS ONE LISTING?

NO

DO YOU WANT A LOADER TAPE PUNCHED?

NO

DO YOU WANT TO SAVE A COPY OF THE MEMORY FOR THE SIMULATOR?

NO

DO YOU WANT A SYMBOL TABLE(SY) OR A CROSS REFERENCE LIST(CR)?

SY
```

Typical PDP11 Operating System Commands

```
$ LOG    100 100                       log onto system
$ DA     08-JAN-76
$ AS     SOURCE, 1                     user program source code
$ AS     SY: LISTING, 5                pass 2 listing file
$ AS     SY: LOADER, 7                 resetting loader tape of user program
$ RUN    ALPS 11                       start the Assembler
```

APPENDIX C

Cross-Assembler Listing

```
GIM LTD ALPS CROSS-ASSEMBLER PASS-ONE ERRORS 13-DEC-71 PAGE 1
    5 QWERT YUIOPLKJH
OPERAND IDENTIFIER TOO LONG
    5 FRED: MODUL PETER
INVALID OPERAND
    7 PETER: EQU SAM
INVALID OPERAND
END OF TAPE 1 FOUND
PLEASE LOAD NEXT TAPE AND THEN TYPE CO
CO
END OF TAPE 2 FOUND
PLEASE LOAD NEXT TAPE AND THEN TYPE CO
CO
    3 ERRORS DETECTED IN PASS ONE

GIM LTD ALPS CROSS-ASSEMBLER INSTRUCTION LISTING 13-DEC-71 PAGE 2
ADR  CODE  ARG  ST-NR            STATEMENT
                  1              ORG    1
                  2   # THIS IS A COMMENT
ØØ1   81    1     3              LAR    1
ØØ2   8Ø          4              LAR    FRED
UNDEFINED OPERAND
ØØ3   ØØ          5              QWERT  YUIOPLK  JH
UNDEFINED OPERATOR
UNDEFINED OPERAND
                  6   FRED       MODUL  PETER
                  7   PETER      EQU    SAM
ØØ4   37    7     8   SAM        SAM    7
ØØ5   4ØØØ        9              JMP    BRIAN
UNDEFINED OPERAND
ØØ7   1Ø          1Ø             HLT                COMMENT
                  11             EOT
                  12  # COMMENT TAPE 2
                  13             EOT
                  14  # TAPE 3
ØØ8   ØF1Ø  1Ø    15             CMP    /FØ
                  16             END

GIM LTD ALPS CROSS-ASSEMBLER SYMBOL TABLE 13-DEC-71 PAGE 3
SYMBOL     ADDR     SYMBOL     ADDR
 SAM       ØØØ4
4 ERRORS DETECTED IN PASS TWO
```

APPENDIX D

Hexadecimal and Modified Hexadecimal

In addition to the decimal and hexadecimal numbers in the source programs the Assembler uses a modified hexadecimal system for the loader tape. In character format, one character is used to represent each ½ word (4 bits), to simplify the loader program, the characters are chosen so that the bottom four bits of their ASCII code are the four bits of the ½ word they represent.

Hexadecimal Number	ASCII Code	Modified Hexadecimal	ASCII Code	Half Word Represented
Ø	00 110 000	Ø	00 110 000	0000
1	00 110 001	1	00 110 001	0001
2	00 110 010	2	00 110 010	0010
3	00 110 011	3	00 110 011	0011
4	00 110 100	4	00 110 100	0100
5	00 110 101	5	00 110 101	0101
6	00 110 110	6	00 110 110	0110
7	00 110 111	7	00 110 111	0111
8	00 111 000	8	00 111 000	1000
9	00 111 001	9	00 111 001	1001
A	01 000 001	J	01 001 010	1010
B	01 000 010	K	01 001 011	1011
C	01 000 011	L	01 001 100	1100
D	01 000 100	M	01 001 101	1101
E	01 000 101	N	01 001 110	1110
F	01 000 110	O	01 001 111	1111

Note: Normally teletypes portray number 0 as Ø and letter O as O

APPENDIX E

ASSEMBLY LANGUAGE

Special Characters

Character	Name	Function
Carriage return		formatting character
line feed		statement terminator
form feed		statement terminator
:	colon	label terminator
#	number sign	in column 1 indicates comment
,	comma	null operand field indicator
	space	field terminator
	tab	field terminator
/	slash	precedes a hexadecimal constant
*	asterisk	current location counter
+	plus sign	arithmetic addition operator, or autoincrement (indirect register mode)
-	minus sign	arithmetic subtraction operator, or autodecrement (indirect register mode)
@	at sign	indirect register mode
'	single quote	ASCII string delimiter
.	period	no present function,reserved for macro facility.
$	dollar sign	no present function,reserved for system use.

SECTION B

THE SIMULATOR

INDEX OF HEADINGS

Chapter One

The ALPS Simulator

1.1 Introduction

The Simulator is a computer program which executes, under controlled conditions, a program written for the ALPS microprocessor. The simulator is written largely in FORTRAN IV, with a few small routines written in Assembly Language. It is designed to be used with the CROSS-ASSEMBLER.

1.2 Configuration of the Simulated System

The hardware of the simulated ALPS system is represented by software and is in a fixed configuration. The configuration represented by the current version of the simulator cannot be altered except by major program changes.

The configuration is:

one processor (LP)

4K of eight bit word Program memory (PM) occupying module addresses 0-15

one module (128 words) of Data Storage (DS), module address 16

eight indirectly accessed peripheral channels, module addresses 48-55

eight directly accessed peripheral channels, module addresses 56-63

See Figure 1.2.1

1.3 Facilities available with the Simulator

The simulator provides program debugging facilities such as instruction tracing, error traps and software generated input signals. The latter facility is an important prerequesite of repeatability of results. Out of bound addresses, illegal instruction words and attempted execution in data areas are some of the errors that can be trapped by the simulator.

1.4 Program Representation and Simulation

The program to be simulated is held in a software representation of two program memory chips, i.e. as a 4K array. The array is generated by the Cross-Assembler and is passed to the Simulator as the simulator file. The Cross-Assembler assembles the instructions of the source program into eight bit words which it stores in the array. The simulator accesses the elements of the array via a software program counter and stack, and decodes the instruction op code. It then "executes" the instruction by making appropriate changes to the variables that represent the hardware. The simulator contains assembly language routines, which access data areas set up by the FORTRAN routines, it is thus very important that both the assembler and simulator are compiled to produce one word (16 bit) integers.

1.5 Instruction Tracing

The trace system used in the simulator allows the user to "mark" areas of code that he wishes to have traced. When the simulator is simulating these areas the states of the accumulator, registers and flags are listed on the line printer before each instruction is simulated. The user can override the trace marks, inhibiting tracing only, by setting switch 0 on the switch register to one. Returning the switbh to zero, restarts tracing.

A special lineprinter driver, written in assembly language, is used to increase the listing speed. The current version (V001A) of the simulator simulates at one thousand'th of the real ALPS processing speed, and at a significantly lower speed when tracing. The areas to be traced are defined before start of simulation when the simulator reads the trace file, which is on logical input/output channel number seven on the host computer. The format of the input is:

TRACE XXX YYY

/*

Where XXX and YYY are hexadecimal addresses marking the start and end of trace areas. Single instructions can be traced by making XXX equal to YYY. When an "end-trace" marker is found, the simulator traces the instruction and thenprints the contents of the processor registers 17-48. Thus the contents of all the registers can be determined at any place in the program by switching the trace off and immediately back on again, i.e. TRACE XXX YYY

```
TRACE YYY ZZZ
```

will cause tracing from XXX to ZZZ with all RAM registers listed at YYY.

Any number of trace cards can appear in the trace file, the /* being treated as an end-of-file marker.

The start and end of trace addresses must lie within the 4K program area and must not contain data items. When a "start trace" marker is found the trace system will be switched on and will remain on until an "end trace" marker is found (and vice versa). Thus leaving the middle of a traced area will not switch off the trace nor will entering the middle of a traced area switch it back on.

To reduce the amount of instruction decoding needed to generate the trace listing the cross-assembler packs a six bit pointer with the 8 bit instruction word into an array element. This pointer points to an element of the instruction mnemonic table. It is removed from the array element in preparation for listing.

1.6 Input and Output

Input signals to the program can be handled by a special subroutine PERIF, which takes as arguments the module number of the I/O channel and an integer constant, and returns a value to the accumulator. PERIF can also be used for output and the integer constant decides whether the operation is input or output.

PERIF must be specially prepared by the user to simulate the input/output characteristics of the peripheral devices attached to the channels. It is called by the simulator whenever a direct (LAM, SAM) or an indirect (LIX, SIX, LIY) input/output instruction is encountered. The format of the call is:

```
CALL PERIF (IPER, IDATA, IO, CLOCK)
where IPER is the module number  (address of I/O port)
      IDATA is the data
      IO is the input output switch    (=Ø for output
                                         =1 for input)
      CLOCK is the value of the simulator cycle counter upon entry
            to PERIF (used for I/O timing)
```

IPER, IDATA and IO are integer variables, and CLOCK is a real number.

1.7 Operating Procedure

The simulator must be supplied with a simulator file and a trace file. The simulator file is read from logical channel number three and the trace file from channel number seven. The simulator identifies itself on the teletype, reads the files and immediately starts simulating. It is thus important to ensure that switch zero on the switch register is set to zero before running the simulator.

When the simulator finds a HALT instruction it terminates tracing, prints the contents of the CPU registers and then stops, printing the message "END OF SIMULATION". If there is no HALT at the end of the program the simulator will fetch the next instruction which will be a return from subroutine (code 00000000) since the assembler zero's all locations before assembly. Thus the simulator will "pop" the subroutine stack causing a jump to some probably undefined address. It is therefore a wise precaution to end a program with a HALT instruction or a "jump to self".

1.8 Interpretation of Output Listing

Before each traced instruction is simulated the simulator lists the state of the accumulators, registers and flags upon the lineprinter. First in the line is the address of the instruction being simulated - this is given as a single hexadecimal digit (the module number), a space and then two hexadecimal digits representing the intra-module address. Next the simulator prints the instruction mnemonic and operand in symbolic form. The contents of the accumulator are printed next as an eight character string of 0's and 1's representing the state of each bit. This is the state of the accumulator before commencement of simulation of the instruction. Similarly the states of the cycle counter, flags, registers and peripheral channels are printed before instruction simulation. After the accumulator contents, the contents of registers T and S are printed. The three flags carry (C), zero (Z) and positive (P) are printed followed by the cycle counter. Next the contents of registers 0-15 are printed, each one being printed as two hexadecimal digits. Finally the eight directly accessible peripheral channels (56-63) are listed, again as two hexadecimal characters. See Figure 1.8.1.

S E C T I O N   C

USING THE ALPS CROSS - ASSEMBLER

ON CYBERNET TIMESHARING LIMITED

USING THE ALPS CROSS-ASSEMBLER ON CYBERNET TIMESHARING LIMITED

In order to use the cross-assembler available from Cybernet Timesharing users require a terminal (like a typewriter) and access to a standard telephone line. To connect to the computer an account code is required, this being available from Cybernet at the address shown below. By dialling one of the telephone numbers supplied by Cybernet, users connect directly with the computers, and after supplying their correct account code they obtain an exclamation mark '!' prompt.

```
CRC INFORMATION SYSTEMS
 14/01/76 10:35
 LINE 03/48
LOGIN: UN,ACCOUNT,PASSWORD
ID=L  SIGMA A  COSMOS 1.2/38

!
```

Note that all output from the computer is underlined.
UN = User name.
ACCOUNT = Account name.
PASSWORD is not normally printed.

Upon receiving the prompt '!' users may enter their assembly programs by BUILDing a file. Programs may be typed in directly or entered from a previously prepared paper tape.

See example over.

Having entered the program(s) in this way the cross-assembler is activated by typing :-

```
!CALL ALPS
```

ALPS then asks for the name of the file which contains the program by prompting :-

```
FILENAME?
```

Here the user types the name of the file specified when building.

EXAMPLE OF FILE BUILDING.

```
!BUILD ALPDEM
   1.000 MODE:EQU 1
   2.000 TIME:EQU 2
   3.000 CHAN :EQU 3
   4.000 RC1:EQU 4
   5.000 RC2:EQU 5
   6.000 RC3:EQU 6
   7.000 COPY:EQU 7
   8.000 RESET:LAL 128
   9.000 SAM 5
  10.000 GOS LOOK
  11.000 LAR COPY
  12.000 SAR MODE
  13.000 LAL 64
  14.000 SAM 5
  15.000 GOS LOOK
  16.000 LAR COPY
  17.000 SAR TIME
  18.000 LAL 32
  19.000 SAM 5
  20.000 COS LOOK
  21.000 LAR COPY
  22.000 ALA 12
  23.000 SAR CHAN
  24.000 LAL 16
  25.000 SAM 5
  26.000 START:LAM 7
  27.000 JIZ START
  28.000 GOS W10
  29.000 LAM 7
  30.000 ALL 1
  31.000 JIZ START
  32.000 LAS 0
  33.000 SAM 5
  34.000 LAR MODE
  35.000 EOL /FF
  36.000 JIP ONCE
  37.000 LSA
  38.000 JIP TEN
  39.000 LSA
  40.000 JIP REP
  41.000 JMP RESET
  42.000 ONCE:GOS TIM
  43.000 JMP FIN
  44.000 TEN:LAS 10
  45.000 SAR MODE
  46.000 ROU:GOS TIM
  47.000 DEC MODE
  48.000 JNZ ROU
  49.000 JMP FIN
  50.000 REP:GOS TIM
  51.000 LAM 7
  52.000 JIZ REP
  53.000 GOS W10
  54.000 LAM 7
  55.000 ALL 2
  56.000 JIZ REP
  57.000 X:LAS 2
  58.000 SAM 4
  59.000 LAM 7
```

ALPDEM is the name of the disc file to be created.

```
  60.000 JNZ X
  61.000 TOOK:LAM 7
  62.000 JIZ TOOK
  63.000 GOS W10
  64.000 LAM 7
  65.000 JIZ TOOK
  66.000 ALL 1
  67.000 JIZ TOOK
  68.000 JMP REP
  69.000 FIN:LAS 1
  70.000 SAM 4
  71.000 JMP FIN
  72.000 LOOJ:LAM 7
  73.000 JIZ LOOK
  74.000 GOS W10
  75.000 LAM 7
  76.000 JIZ LOOK
  77.000 SAR COPY
  78.000 LAS 0
  79.000 SAM 5
  80.000 NOT:LAM 7
  81.000 JNZ NOT
  82.000 RET
  83.000 W10:LAL 255
  84.000 SAR RC1
  85.000 LOOP: DEC RC1
  86.000 JNZ LOOP
  87.000 RET
  88.000 TIM:GOS COUNT
  89.000 LAR CHAN
  90.000 SAM 4
  91.000 GOS COUNT
  92.000 KAS 0
  93.000 SAM 4
  94.000 RET
  95.000 COUNT:LAR TIME
  96.000 SAR RC3
  97.000 2ND:LAL 255
  98.000 SAR RC2
  99.000 1ST:DEC RC2
 100.000 JNZ 1ST
 101.000 DEC RC3
 102.000 JNZ 2ND
 103.000 RET
 104.000 END
```

Next the program prompts for options :-

OPTIONS?

If no options are required then a carriage return causes ALPS to compile the assembly program using standard defaults. (See below.) If the user wishes to know what options are available then a question mark should be typed in response to the OPTIONS? request.

e.g.
```
!CALL ALPS
 GIM LTD ALPS CROSS-ASSEMBLER VCRC002 14/ 1/76
FILENAME:ALPDEM
OPTIONS:?
OPTIONS ARE,

 P1 (BATCH DEFAULT)  PASS 1 LISTING STARTING
 P1(N)               AT STATEMENT  NO. N
 N1 (T/S DEFAULT)    NO PASS 1 LISTING

 P2 (BATCH DEFAULT)  PASS 2 LISTING
 N2 (T/S DEFAULT)    NO PASS 2 LISTING

 CR (BATCH DEFAULT)  CROSS REFERENCE LISTING
 NCR (T/S DEFAULT)   NO CROSS REFERENCE LISTING

 SY                  SYMBOL TABLE

 SI                  MEMORY DUMP FOR SIMULATOR VIA. F:3

 LOS(FILE) (T/S DEFAULT TO LOTEMP(ID))
                     LOADER TAPE IN MODIFIED HEX TO "FILE"
 LOL(FILE)           LOADER TAPE IN STANDARD HEX TO "FILE"
 NL (BATCH DEFAULT)  NO LOADER TAPE
OPTIONS:[RET]
 003  7800          10            GOS      LOOK
ERROR ON LINE 10
 UNDEFINED OPERAND
 00A  7800          15            GOS      LOOK
ERROR ON LINE 15
 UNDEFINED OPERAND
 011    00          20            COS      LOOK
ERROR ON LINE 20
 UNDEFINED OPERATOR
ERROR ON LINE 20
 UNDEFINED OPERAND
 063  4800          73            JIZ      LOOK
ERROR ON LINE 73
 UNDEFINED OPERAND
 068  4800          76            JIZ      LOOK
ERROR ON LINE 76
 UNDEFINED OPERAND
 07E    00          92            KAS      0
ERROR ON LINE 92
 UNDEFINED OPERATOR
 7 ERRORS DETECTED IN PASS TWO

 *EXIT*
```

N.B. Options should be entered on one line, separated by commas.

The assembler performs two passes through the assembly program checking first for syntax errors and then for references to labels and jump instructions. No listings of the program are produced in either pass 1 or pass 2 unless requested by specifying the options P1 or P2 (this saves time in listing at users terminal). Diagnostic error messages are supplied along with an indication of the line on which the error has occured. If no errors are detected then a loader tape file is produced and this may be punched at the user's terminal if required.

Errors may be corrected by editing the assembly program file :-

```
!EDIT ALPDEM
*TY10                                   Type line 10.000
  10.000 GOS LOOK
*TY15                                   Type line 15.000
  15.000 GOS LOOK
*TY20                                   Type line 20.000
  20.000 COS LOOK
*OV20                                   Overwrite line 20.000
  20.000 GOS LOOK
*TY73
  73.000 JIZ LOOK
*TY76
  76.000 JIZ LOOK
*TY92
  92.000 KAS 0
*OV92
  92.000 LAS 0
*TY72
  72.000 LOOJ:LAM 7
*OV72
  72.000 LOOK:LAM 7
*X                                      X - leave the editor.
!CALL ALPS
 GIM LTD ALPS CROSS-ASSEMBLER VCRC002 14/ 1/76
FILENAME:ALPDEM
OPTIONS:[RET]

 ** NO ERRORS FLAGGED IN ABOVE ASSEMBLY **
    LOADER TAPE IN FILE:- LOTEMPL
  *EXIT*

!PUNCH LOTEMPL
PLEASE TURN ON YOUR PUNCH AND PRESS [RET]
|04 80 35 78 63 87 91 04 40 35 78 63 87 92 04 20   Note that the tape is
|35 78 63 87 0N 0L 93 04 10 35 27 48 1J 78 72 27   produced in modified
|05 01 48 1J 00 35 81 0L 00 58 33 1L 58 37 1L 58   Hexadecimal. For stan-
|40 40 00 78 79 40 50 0J 91 78 79 M1 50 39 40 50   dard Hexadecimal see
|78 79 27 48 40 78 72 27 05 02 48 40 02 34 27 50   option LOL.
|4L 27 48 51 78 72 27 48 51 05 01 48 51 40 40 01
|34 40 50 27 48 63 78 72 27 48 63 97 00 35 27 50
|6N 00 04 00 94 M4 50 75 00 78 82 83 34 78 82 00
|34 00 82 96 04 00 95 M5 50 87 M6 50 84 00
|?05KN
```

As may be seen from the above example, to edit a file the user must type 'EDIT filename' after a '!' prompt.

```
!EDIT filename
*               Asterisk is the edit subsystem prompt.
```

Note that when editing an asterisk '*' prompt is obtained. There are numerous commands available within the editor but only four are required for most operations relating to ALPS. These are :-

TYn	Type the line number n
DEn	Delete line number n
INn	Insert line number n
OVn	Overwrite line number n

To leave the editor users simply type X [RET] in response to the '*' prompt.

When users have obtained a correct assembly program they may obtain a pass one and pass two listing by running the ALPS program in a batch mode where the printout is obtained on a line printer and the loader tape is produced on a high speed paper tape punch. To perform this operation all that is required is to type :

```
!CALL FROM ALPF
```

This results in a series of questions which will instigate the job. If an explanation of any question asked is required a '?' reply will provide this.

Note / may be used in place of 'CALL'
A question mark response produces an explanation.

```
!/FROM ALPF
NAME OF PROGRAM FILE:?
ENTER HERE THE NAME OF THE FILE WHICH CONTAINS THE
MNEMONIC ASSEMBLY PROGRAM FOR THE LP8000
NAME OF PROGRAM FILE:ALPDEM
!JOBQ
```

JOBQ produces information about the BATCH queue.

```
12:21 CURRENT JOB (ID=1065) 34 MINS (LIMIT=200 MINS)
 1 AT PRIORITY C
 3 AT PRIORITY 7
12 AT PRIORITY 2
 6 AT PRIORITY 0
WHICH BATCH PRIORITY IS REQUIRED??
BATCH PRIORITIES ARE:-
C FOR IMMEDIATE RUN @ 5.30 PER C.U.
B FOR HIGH PRIORITY @ 3.50 PER C.U.
A FOR STD. PRIORITY @ 2.90 PER C.U.
7 FOR LOW  PRIORITY @ 2.50 PER C.U.
2 FOR O'NT PRIORITY @ 2.10 PER C.U.

USERS SHOULD CHOOSE PRIORITIES IN THIS RANGE
WHICH BATCH PRIORITY IS REQUIRED?7
```

Priority 7 is low priority.

```
TIME LIMIT:?
THE TIME LIMIT REQUIRED IS FOR THE RUNNING OF THE JOB
THIS WILL DEPEND ON THE SIZE OF THE PROGRAM BUT A
JUDICIOUS VALUE WOULD BE 10 MINUTES.
TIME LIMIT:10
LOADER TAPE FILE NAME:?
THIS IS THE NAME OF THE FILE THAT THE PROGRAM WILL
CREATE AND PUNCH WHICH CONTAINS THE ASSEMBLED PROGRAM.
LOADER TAPE FILE NAME:ALPDTAPE
NAME & ADDRESS: :?
THIS IS THE NAME AND ADDRESS TO WHICH THE OUTPUT AND
PAPER TAPE IS TO BE SENT.
TERMINATE THE ENTRY BY TYPING A CARRIAGE RETURN
I.E. A BLANK LINE
NAME & ADDRESS: :O/P TO COLIN HAVERCROFT
:83 CLERKENWEEL ←←←LL RD.
:   LONDON   EC1.
:>
INSERT JOB?Y
```

Y or YES may be given.

```
JOB INSERTED.ID=1079
TAPE IN FILE ALPDTAPE
JOB IN FILE SOTEMP4
```

EXAMPLE SHOWING NORMAL USAGE WITHOUT THE QUESTION MARK RESPONSE.

```
!/FROM ALPF
NAME OF PROGRAM FILE:ALPDEM
!JOBQ
12:25 CURRENT JOB (ID=1072) 2 MINS (LIMIT=45 MINS)
 6 AT PRIORITY 7
16 AT PRIORITY 2
 6 AT PRIORITY 0
WHICH BATCH PRIORITY IS REQUIRED?7
TIME LIMIT:10
LOADER TAPE FILE NAME:ALPDTAPE
NAME & ADDRESS: :O/P TO C.HAVERCROFT AT 83 PLEASE.
:
INSERT JOB?N
JOB IN FILE SOTEMP4

!
```

Additional Features of Cybernet ALPS

Direct program entry

If it is required to enter small programs or to test assembler instructions to discover their equivalent hexadecimal code, the facility exists for typing assembler instructions directly into the ALPS program. The disadvantage of this method is that no permanent record of the entered statements is created i.e. the statements entered are deleted when leaving ALPS, and are not stored on disc file as when using BUILD.

e.g.

```
 !CALL ALPS
 GIM LTD ALPS CROSS-ASSEMBLER VCRC002 14/ 1/76
FILENAME:[RET]
OPTIONS:P2
 TYPE IN PROGRAM NOW
?ASCII 'THE QUICK BROW'
?END
 GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   14/ 1/76 PAGE   1
 ADR  CODE   ARG  ST-NR         STATEMENT

 000   54    'T'    1           ASCII  'THE QUICK BROW'
 001   48    'H'
 002   45    'E'
 003   20    ' '
 004   51    'Q'
 005   55    'U'
 006   49    'I'
 007   43    'C'
 008   4B    'K'
 009   20    ' '
 00A   42    'B'
 00B   52    'R'
 00C   4F    'O'
 00D   57    'W'
                    2           END

 ** NO ERRORS FLAGGED IN ABOVE ASSEMBLY **
   LOADER TAPE IN FILE:- LOTEMPL
*EXIT*
```

LOS and LOL

The options LOS and LOL allow the user to specify whether the loader tape hexadecimal notation produced is in modified or in standard format. By default (i. e. if no option is specified) the program will produce modified hexadecimal as required by the General Instruments LP8000 proto-type kit. If standard hexadecimal is required the LOL option should be specified.

When ALPS is required to generate a specific loader tape file (and not the default file LOTEMP) users should specify either :-

LOS (filename)	-	for modified hexadecimal
LOL (filename)	-	for standard hexadecimal

e.g.

```
LOADER TAPE SHOWN HERE IN MODIFIED HEXADECIMAL

04 80 35 78 63 87 91 04 40 35 78 63 87 92 04 20
35 78 63 87 0N 0L 93 04 10 35 27 48 1J 78 72 27
05 01 48 1J 00 35 81 0L 00 58 33 1L 58 37 1L 58
40 40 00 78 79 40 50 0J 91 78 79 M1 50 39 40 50
78 79 27 48 40 78 72 27 05 02 48 40 02 34 27 50
4L 27 48 51 78 72 27 48 51 05 01 48 51 40 40 01
34 40 50 27 48 63 78 72 27 48 63 97 00 35 27 50
6N 00 04 00 94 M4 50 75 00 78 82 83 34 78 82 00
34 00 82 96 04 00 95 M5 50 87 M6 50 84 00
205KN
```

```
LOADER TAPE SHOWN HERE IN STANDARD HEXADECIMAL

04 80 35 78 63 87 91 04 40 35 78 63 87 92 04 20
35 78 63 87 0E 0C 93 04 10 35 27 48 1A 78 72 27
05 01 48 1A F0 35 81 0C FF 58 33 1C 58 37 1C 58
40 40 00 78 79 40 5F FA 91 78 79 D1 50 39 40 5F
78 79 27 48 40 78 72 27 05 02 48 40 F2 34 27 50
4C 27 48 51 78 72 27 48 51 05 01 48 51 40 40 F1
34 40 5F 27 48 63 78 72 27 48 63 97 F0 35 27 50
6E 00 04 FF 94 D4 50 75 00 78 82 83 34 78 82 F0
34 00 82 96 04 FF 95 D5 50 87 D6 50 84 00
205BE
```

EOT

The assembly instruction EOT (end of tape) has been modified to allow users to break up their programs into several smaller files. Each of these individual files may be compiled separately and should have EOT as the last assembly instruction in place of the END statement.

When ALPS encounters the EOT instruction it issues the message :-

```
          END OF FILE XXXXX        FOUND
          PLEASE ENTER NEXT FILENAME
      FILENAME:
```

where XXXXX is the filename

The next file should be specified and the pass 1 assembly will continue. If it is required to terminate the first pass a carriage return should be given in response to the FILENAME: request and this will allow entry of either further lines of assembler typed directly at the terminal or to supply an END instruction which will cause ALPS to proceed to the pass 2 assembly.

e.g.

```
!BUILD TIM                             !BUILD COUNT
   1.000 TIM:GOS COUNT                    1.000 COUNT:LAR TIME
   2.000 LAR CHAN                         2.000 SAR RC3
   3.000 SAM 4                            3.000 2ND:LAL 255
   4.000 GOS COUNT                        4.000 SAR RC2
   5.000 KAS 0                            5.000 1ST:DEC RC2
   6.000 SAM 4                            6.000 JNZ 1ST
   7.000 RET                              7.000 DEC RC3
   8.000 EOT                              8.000 JNZ 2ND
   9.000 [RET]                            9.000 RET
*X                                       10.000 EOT
                                         11.000 [RET]
                                       *X
```

The two subroutines shown above have been built into separate files, they have been extracted from the example shown previously.

```
!CALL ALPS
 GIM LTD ALPS CROSS-ASSEMBLER VCRC00212/ 1/76
FILENAME:TIM
OPTIONS:[RET]

     END OF FILE TIM             FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:COUNT

     END OF FILE COUNT           FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:[RET]
TYPE IN THE PROGRAM NOW
?END
 002    80          2                LAR      CHAN
ERROR ON LINE 2
 UNDEFINED OPERAND
 006    00          5                KAS      0
ERROR ON LINE 5
 UNDEFINED OPERATOR
 009    80          9    COUNT       LAR      TIME
ERROR ON LINE 9
 UNDEFINED OPERAND
 00A    90          10               SAR      RC3
ERROR ON LINE 10
 UNDEFINED OPERAND
 00D    90          12               SAR      RC2
ERROR ON LINE 12
 UNDEFINED OPERAND
 00E    D0          13   1ST         DEC      RC2
ERROR ON LINE 13
 UNDEFINED OPERAND
 011    D0          15               DEC      RC3
ERROR ON LINE 15
 UNDEFINED OPERAND
   7 ERRORS DETECTED IN PASS TWO
*EXIT*
```

Most of the above errors are due to references to labels which are defined in the main program. These routines are assembled with the remainder of the program, which has also been broken down into the main program and separate subroutines, (after corrections to the incorrect syntax on line 5 have been made). The resultant assembly is shown below and a cross reference table is also shown.

```
!CALL ALPS
 GIM LTD ALPS CROSS-ASSEMBLER VCRC002 12/ 1/76
FILENAME:MAIN
OPTIONS:CR
     END OF FILE MAIN           FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:LOOK
     END OF FILE LOOK           FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:W10
     END OF FILE W10            FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:TIM
     END OF FILE TIM            FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:COUNT
     END OF FILE COUNT          FOUND
     PLEASE ENTER NEXT FILENAME
FILENAME:[RET]
TYPE IN THE PROGRAM NOW
?END
 GIM LTD ALPS CROSS-ASSEMBLER        CROSS REFERENCE LIST   12/ 1/76 PAGE 2
 ST-NR     NAME         REFERENCED IN STATEMENT(S)
     3     CHAN             91      23
     7     COPY             77      21      16      11
    98     COUNT            93      90
    69     FIN              49      43
    72     LOOK             76      73      20      15      10
    86     LOOP             87
     1     MODE             47      45      34      12
    80     NOT              81
    42     ONCE             36
     4     RC1              86      85
     5     RC2             102     101
     6     RC3             104      99
    50     REP              68      56      52      40
     8     RESET            41
    46     ROU              48
    26     START            31      27
    44     TEN              38
    90     TIM              50      46      42
     2     TIME             98      17
    61     TOOK             67      65      62
    84     W10              74      63      53      28
    57     X                60
   102     1ST             103
   100     2ND             105
 ** NO ERRORS FLAGGED IN ABOVE ASSEMBLY **
    LOADER TAPE IN FILE:- LOTEMP3
*EXIT*
!
```

SECTION D

PRJGRAMMING EXAMPLES

(See also section 4.3, page 42, of Hardware Manual)

1.0 Simple Input-Output

The I/O ports of the ALPS system are 8 bit wide and contain output latches, which means that when data is sent to them it remains on the output pins until it is overwritten with new data. The same ports can be used to input data but it must be remembered that the input pins are shared with the output pins and so any data read into the processor will be the wire-OR of the current data on the output latches and the current input data. If a port is only used to read in data then it must have its output latches cleared before being used for input, this is done automatically by the power-on-reset circuitry.

DIRECT ADDRESSED I/O

The LAM (Load Accumulator from Module) and SAM (Store Accumulator in Module) instructions can be used to transfer data between the LP8000 accumulator and the I/O ports at module addresses 56 to 63 at the top end of the memory address range. As the LAM and SAM instructions only have 3 bits available to describe the module address of the data port to be used the 3 bit code has 56 added to it by the processor to form the actual module address thus

```
        LAM Ø addresses I/O port 56
        LAM 7 addresses I/O port 63
```

The I/O ports can be on LP6000 ROMs, LP1010 I/O Buffers or the LP8000 CPU chip, the latter port being exceptional in that:

1.1 it always has address 63

1.2 all 8 bits can be used for input but

1.3 only the uppermost four bits can be used for output.

A simple output program could be

```
1.4   LAL DATA
1.5   SAM IOPORT
```

where 1.4 sets the accumulator up with the required 8 bit data and 1.5 outputs the data to the IOPORT, IOPORT being a variable in the range 0 to 7 to specify a module address in the range 56 to 63.

A simple input program could be:

```
1.6   LAM IOPORT
1.7   SAR REG
```

where 1.6 inputs the wire-OR of any previous output and the new input from data port IOPORT, IOPORT being a variable in the range 0 to 7 as in 1.5 above.

INDIRECT ADDRESSING

In the unlikely event that 8 I/O ports will not provide enough I/O for a system the I/O can be extended by indirect addressing. Any of the 64 module addresses can be used as I/O port addresses however it is a convention that I/O addresses start at the top of memory and grow downwards whereas program memory starts at the bottom of memory and grows upwards. The ALPS prototyping system allows the user to access 15 I/O ports with module addresses in the range 49-63.

A simple input or output program might be:

```
1.8   LAL IOADDR
1.9   SAX
1.10  SIX or LIX
```

where 1.8 sets the accumulator to the module address of the I/O port required, 1.9 then stores this value in the X register and 2.0 either stores or loads data between the accumulator and I/O port. Note that once the X register is set up any number of LIX and SIX instructions can be used, the X register only needs to be set up when a different I/O port is required.

Indirect addressed input-output provides more addresses for I/O ports (up to 64) but takes more program and takes a longer time for the instructions to execute and therefore direct addressng should be used whenever possible.

2.0 Zeroing Registers and RAM

Sometimes it is desirable to start running a program knowing that the CPU registers and main memory RAM are cleared of any data.

A program to do this for CPU registers could be:

```
        LSS 7           SET UP 'S' REGISTER
        LTS N           SET UP PAGE OF CPU MEMORY TO BE CLEARED
        LAS Ø           CLEAR ACCUMULATOR
  LOOP: SAR @ -         STORE Ø IN RAM (S,T), S = S - 1
        JRS LOOP        IF S ≠ 7 GO TO "LOOP"
        LTS NEXT        SET UP NEXT PAGE TO BE CLEARED
                        NOTE THAT S = 7 FROM PREVIOUS LOOP
 LOOP2: SAR @ -         STORE Ø IN RAM (S,T), S = S - 1
        JRS LOOP2       IF S ≠ 7 GO TO "LOOP2"
          '             ETC.  ETC.
          '
          '
```

In the above program N and NEXT hold the 'T' value of the CPU register page to be cleared. The value of S is decremented from 7 to Ø in each loop, note that "S"rolls over" from Ø to 7 when decremented again.

Clearing 256 words of RAM in main memory can be done as follows:

```
        LAL MODULE      SET UP MODULE ADDRESS OF 256 WORD BLOCK TO BE CLEARED
        SAX ,           STORE MODULE ADDRESS IN X
        LAS Ø     )     SET UP STARTING VALUE OF LOWER
        SAR REG   )     8 BITS OF ADDRESS
  LOOP: LAR REG         GET 8 BIT ADDRESS IN ACCUM
        SZX ,           STORE RAM ADDRESS IN Z REGISTER OF CHIP SELECTED
                        AS MODULE 'X'
        LAS Ø           CLEAR ACCUMULATOR
        SIX ,           STORE Ø IN RAM
        DEC REG         DECREMENT RAM ADDRESS
        JNZ LOOP        ALL CLEARED ?
```

In this program the RAM locations are cleared in the order Ø, 255 - - - - 1, a CPU register called REG being used to hold the value of the next RAM address to be cleared. Note that advantage is taken of the fact that the DEC decrement instruction changes a value of Ø to a value 255.

3.0 Moving Data Between Registers

DIRECT ADDRESSING

The simplest and quickest way to move data between registers is by:

```
        LAR REG 1
        SAR REG 2
```

which copies the data in REG 1 into REG 2 assuming REG 1 and REG 2 are directly addressed, this infers that only register 0-11 and V, W, X and Y can be used in this way.

INDIRECT ADDRESSING

By using indirect addressing it is possible to transfer data from one page of 8 register to another:

```
        LSS 7            SET UP S = 7
  LOOP: LTS TVALUE       SET UP 'T' VALUE
        LAR @            GET DATA
        LTS Ø            ALTER CPU REGISTER PAGE TO T = Ø
        SAR @-           NOTE THE AUTO DECREMENT OF S
        JRS LOOP         ALL DONE ?
```

The above program copies the 8 registers in CPU page TVALUE into page 0 of the CPU registers. It would be quite easy to alter the above program so that the registers could be copied 'upside down' or copied out to external RAM etc. Further program examples are given in the Standard Routines Section D, item 9.0.

4.0 BCD Addition

The ALPS microprocessor is particularly well suited to BCD arithmetic operations and manipulations and a number of routines are given in the Standard Routines of Section D paragraph 9.0.

The basic method for BCD addition is as follows:

Problem: REG 3 = REG 3 + DATA

```
4.1   LAL DATA
4.2   ALA /66
4.3   ADR REG 3
```

The program works as follows:

Statement 4.1 puts the data value in the accumulator,

Statement 4.2 corrects this number ready for BCD addition, and

Statement 4.3 adds the contents of the accumulator to REG 3 and corrects for BCD carries.

The BCD add instruction adds two BCD numbers in the accumulator to two BCD numbers in a register. Any carry from the first two numbers is added in with the higher significant numbers, any carry resulting from that addition sets the processor carry flag and can be used for testing by conditional jump instructions.

5.0 BCD Subtraction

The BCD subtraction method is very similar to BCD addition as described in the previous paragraph. The basic program is shown below, a more comprehensive example is given in paragraph 9.0, Standard Routines.

Problem REG 3 = REG 3 - DATA

```
        LSA                SET CARRY = 1
        LAR DATA           GET DATA
        EOL /FF            ONES COMPLEMENT IT
        ADR REG 3          ADD IT TO REG 3
```

The main differences between addition and subtraction are:

A. the carry flip flop should be set to a 1 before subtraction
0 before addition

B. the data is inverted for subtraction

C. the data does not need a correcting number /66 added to it for subtraction.

6.0 Changing Pages/Table Jumps

The ALPS microprocessor memory splits naturally into 8 x 2K word pages, jump and subroutine instructions having an eleven bit addressing capability to operate within a 2K page. If there is a requirement to jump from one 2K page to another (e.g. when a program is larger than 2K words) then the following programming technique can be used:

```
6.1   LAL PAGMOD    LOAD PAGE AND MODULE ADDRESS
6.2   SAX           STORE IN X REG
6.3   LAL WORD      LOAD WORD ADDRESS WITHIN MODULE
6.4   SQX           STORE 11 BIT MODULE/WORD ADDRESS IN Q (PAGE)
6.5   LAX           GET PAGE AND MODULE ADDRESS
6.6   SAY           STORE IN Y REGISTER TO SELECT CHIP
6.7   SAY           DUMMY INSTRUCTION FOR OVERLAP
```

In the above program the instructions function as follows:

6.1 the value of PAGMOD is put into the accumulator,
PAGMOD bits 4, 5, 6 specify the new page required
PAGMOD bits 1, 2, 3 specify the module within the new page.

6.2 the value of PAGMOD, the module address part of the new address, is stored in the X register.

6.3 the WORD part (bottom 8 bits) of the new address is put into the accumulator. Note that if this value were Ø, as in the case of setting up the start address of a new ROM, then LAS Ø could be used instead of LAL WORD thus saving one instruction word.

6.4 the SQX instruction stores the contents of the accumulator (WORD) and the lower three bits of the X register contents in the 11 bit Q register of the memory device which has a page address equal to the value of bits 4, 5, 6 of the X register.

6.5 this instruction retrieves the value in the X register and puts it in the accumulator.

6.6 the new memory is now chip selected by storing the new page address (PAGMOD) in the Y register.

6.7 this is a dummy instruction to compensate for the fact that instructions are fetched from memory whilst the previous instruction is being obeyed, ("overlapped fetch"). It has no programming function as such.

7.0 A Keyboard Input and Seven Segment Display Routine

In order to understand this demonstration program it will be necessary to study this explanation in conjunction with flowchart and program listing. The overall function of the program is to continuously examine a 16 key keyboard to check if a key is pressed, if any key is pressed then the program converts the key code to BCD and displays it on 4 LEDs and then converts the BCD code to seven segment code and displays it on a seven segment display. The electrical circuitry required between the keyboard, LEDs and displays is very simple and is shown in the circuit diagram.

The program assumes the keyboard is connected to I/O module 62, the seven segment display is connected to I/O module 61 and the LEDs are connected to I/O module 63 (on the CPU chip). The program functions as follows:

7.1 At label INKB: registers 1 and 2 are set up as counters with an initial value of 4, these counters are used later in the program to help convert the keyboard code to BCD.

7.2 A code of 0000 1111 is then output to the keyboard matrix (see circuit diagram for connections) and then the output of the keyboard matrix is read back in and the top 4 bits checked for a non zero result. If the result is zero no key was pressed so the displays are cleared at NOKB: and the program restarts.

7.3 If the result was non-zero then the newly input keyboard code is sent back out to the keyboard on the top 4 bits of IO62. These 4 bits are shifted to the top end of the accumulator and then the result checked, if the result is zero then no key is pressed or rather the key depression previously detected must have disappeared so the displays are cleared at NOKB: and the program restarted.

7.4 If a key is still pressed then the two co-ordinates X and Y, of the key are now stored in the top and bottom four bits of the keyboard interface (remember the I/O interface consists of latches). The X and Y codes (1 or 4) are now converted to numbers in the range 0-3 by counting down the registers 1 and 2, which have previously been reset to 4, until the relevent key has been accounted for.

7.5/

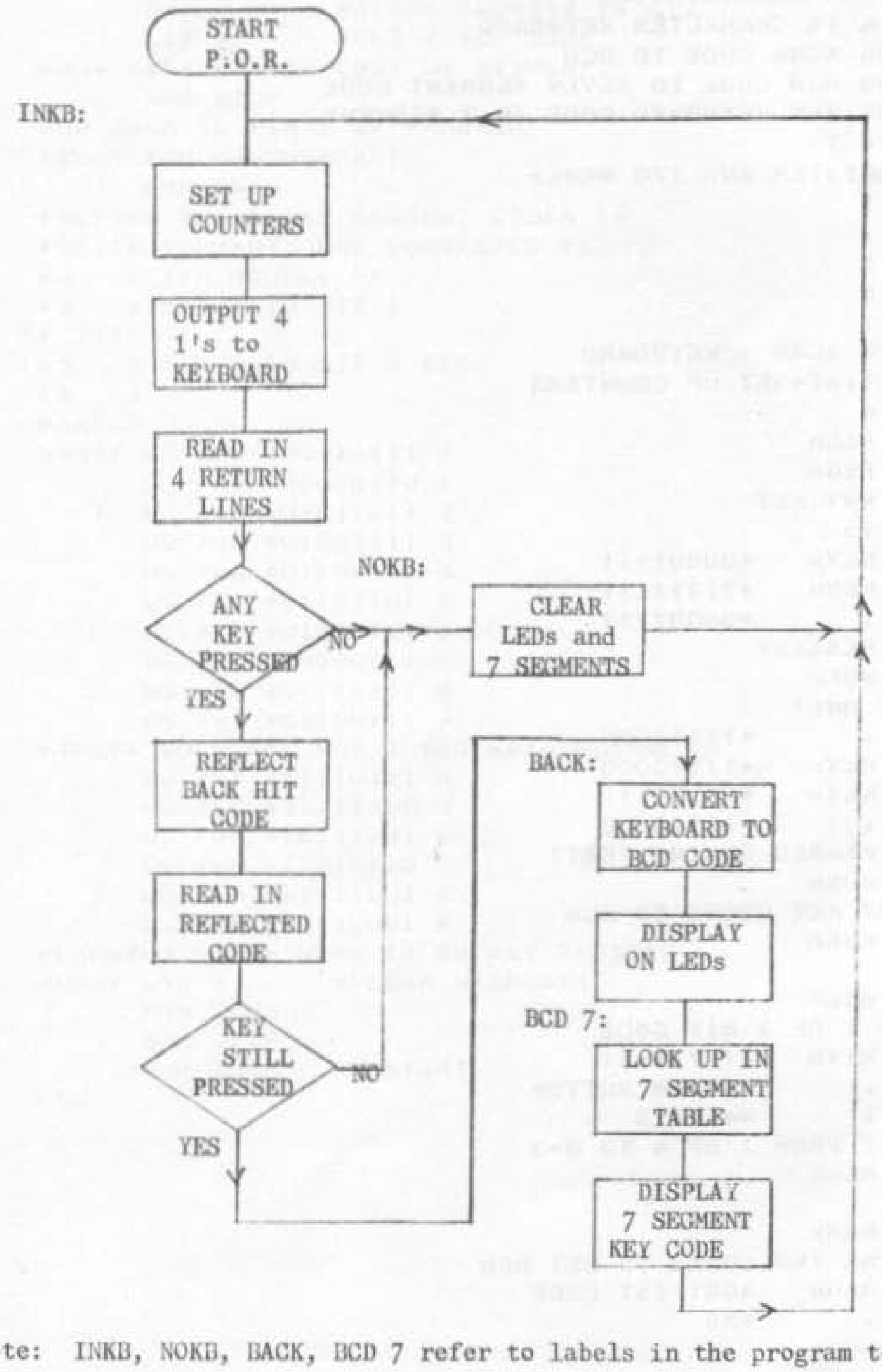
7.5 The codes are then converted to BCD by shifting the one in REGA left 2 places (to multiply it by 4) and then adding it to the other in REGB. This code is then shifted to the top 4 bits of the accumulator and displayed on the CPU I/O register 63 (remember this only has output devices on its top 4 bits).

7.6 The BCD code is restored to the bottom 4 bits of the accumulator and then at label BCD7: it is added to the start address of a table of the 7 segment equivalents of BCD (rather hexadecimal) numbers. This new address is the address of the data defining the 7 segment equivalent of the BCD number.

7.7 This address is now stored in the Z register of the same module as the program (SZY) and then the appropriate 7 segment code is read in from the program ROM via the Z register (LIY). The table of 7 segment codes is defined as part of the program at label SEG 7:, note that these binary codes are never executed as program because of the JMP INKB instruction preceeding them and the fact that the label SEG 7: is never used as the destination of a jump instruction.

A flowchart describing this program is shown in figure 7.8 and some notes on the electrical circuit required is given in figure 7.9.

ALPS-LP8000 KEYBOARD-DISPLAY PROGRAM EXAMPLE



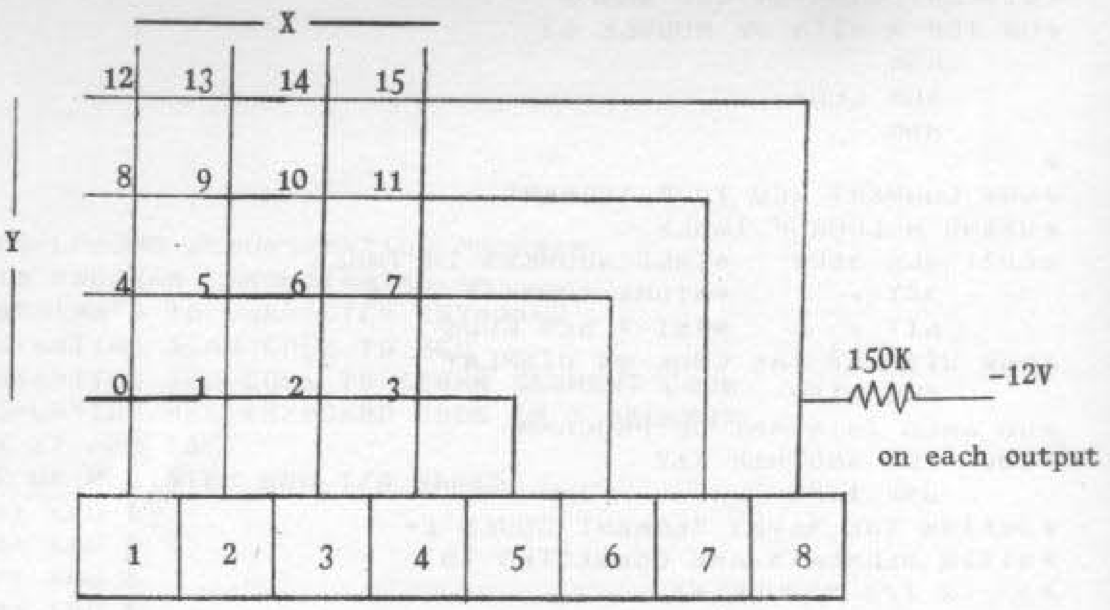
Note: INKB, NOKB, BACK, BCD 7 refer to labels in the program text.

Figure 7.8

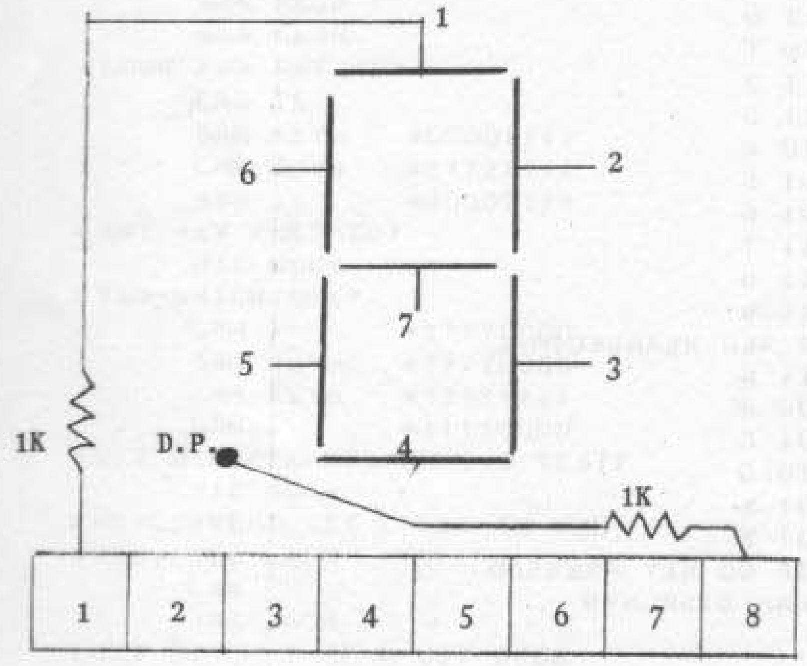
```
#ALPS-LP8000 DEMONSTRATION PROGRAM
#THIS PROGRAM DEMONSTRATES :-
#SCANNING A 16 CHARACTER KEYBOARD
#CONVERTING SCAN CODE TO BCD
#CONVERTING BCD CODE TO SEVEN SEGMENT CODE
#DISPLAYING HEX KEYBOARD CODE IN 7 SEGMENT
#BRK 27 APR 75
#SET UP REGISTER AND I/O NAMES
REGA: EQU 1
REGB: EQU 2
DISP: EQU 5
KEYB: EQU 6
LEDS: EQU 7
#ROUTINE TO SCAN A KEYBOARD
#START ROUTINE-SET UP COUNTERS
INKB: LAS 4
      SAR REGA
      SAR REGB
#LOOK FOR ANY KEY
      LAS 15
      SAM KEYB    *0000!!!!
      LAM KEYB    *????!!!!
      RSN ,       *0000????
#ANY KEY PRESSED?
      JIZ NOKB
#YES-WHICH ONE?
      LSN ,       *????0000
      SAM KEYB    *????0000
      LAM KEYB    *????!!!!
      LSN ,       *!!!!0000
#KEY DISAPPEARED DURING TEST?
      JIZ NOKB
#NO-CONVERT KEY CODES TO BCD
BACK: DEC REGB
      LSA
      JNZ BACK
#GET OTHER 1 OF 4 BIT CODE
      LAM KEYB    *????!!!!
      RSN ,       *CLEAR BOTTOM
      LSN ,       *4 BITS
#CONVERT IT FROM 1 OF 4 TO 0-3
MORE: DEC REGA
      LSA
      JNZ MORE
#COMBINE THE TWO CODES TO GET BCD
      LAR REGA    *GET 1ST CODE
      LSA ,       *X4
      LSA
      BAD REGB
#HEXADECIMAL RESULT IN ACCUMULATOR
```

```
#
#DISPLAY CHAR ON LED LAMPS
#ON TOP 4 BITS OF MODULE 63
        LSN
        SAM LEDS
        RSN
#
#NOW CONVERT BCD TO 7 SEGMENT
#USING A LOOKUP TABLE
BCD7: ALA SEG7      *CALC ADDRESS IN TABLE
        SZY ,       *STORE ADDRESS IN Z
        LIY ,       *GET 7 SEG CODE
#NOW DISPLAY THE CODE ON DISPLAY
        SAM DISP
#GO BACK TO START OF PROGRAM
#LOOK FOR ANOTHER KEY
        JMP INKB
#DEFINE THE SEVEN SEGMENT CODES :-
#61112 SEGMENTS ARE CONNECTED TO
#6   2 I/O MODULE 61
#6   2 SEG 1 TO BIT 1
# 777
#5   3 SEG 2 TO BIT 2 ETC.
#5   3
#54443
SEG7: DC /3F *00111111 0
        DC /06 *00000110 1
        DC /5B *01011011 2
        DC /4F *01001111 3
        DC /66 *01100110 4
        DC /6D *01101101 5
        DC /7D *01111101 6
        DC /07 *00000111 7
        DC /7F *01111111 8
        DC /67 *01100111 9
#THESE CODE ARE FOR 7 SEG HEXADECIMAL
        DC /F7 *11110111 A
        DC /FC *11111100 B
        DC /B9 *10111001 C
        DC /DE *11011110 D
        DC /F9 *11111001 E
        DC /F1 *11110001 F
#PROGRAM COMES HERE IF NO KEY PRESSED
NOKB: LAS 0         *CLEAR DISPLAYS
        SAM LEDS
        SAM DISP
        JMP INKB    *RESTART
END
```

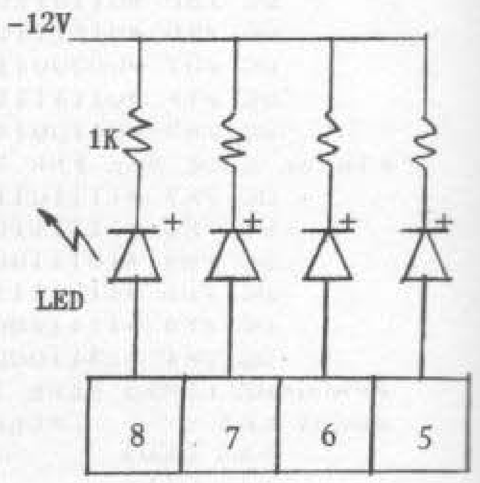
ELECTRICAL DETAILS   Keyboard/7 Segment Display



IO 62 Keyboard



IO 61 7 Segment Display
(common cathode to -12V)



IO 63 LEDS

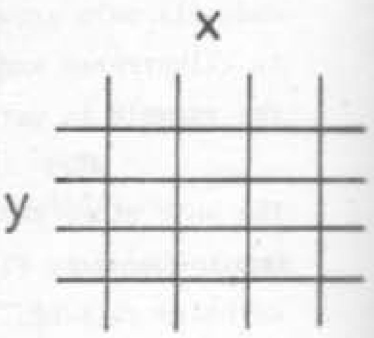
Figure 7.9

8.0 A Second Keyboard Program

The keyboard routine described here is more sophisticated than the one used in paragraph 8.0. It can detect when one or more keys are depressed simultaneously and will only accept key inputs if one key is depressed at a time. The program is illustrated and explained in Figure 8.1 and should be simple to follow once the example in paragraph 8.0 has been understood.

The only other keyboard facility that would be particularly desirable that is not included in Figure 8.1 is that of anti-contact bounce logic for the keyboard switches. This facility can easily be added by programming a short timer (5mS) using a loop of instructions and then using the keyboard routine again. If the keyboard routine detects the same key twice then it can be assumed that the key contact is not bouncing. For this reason (using the keyboard routine twice) it is usual to program the keyboard routine as a subroutine that is called from the main program.

ALTERNATIVE KEYBOARD SERVICE ROUTINE



```
LAL STRTAB         } SETTING UP TABLE MODULE
SAX                } ADDRESS FOR LATER ACCESS
LAS 15             }
                   } SET UP 1111 on X
SAM X              }
LAM Y                READ IN ON Y
JIZ NOKEY
SAM Y                ECHO BACK ON Y
LAS Ø              }
                   } CLEAR PORT X
SAM X              }
LAM X                READ IN ON X
ALA STRTAB           ADDING INDEX OBTAINED FROM X TO START
                     ADDRESS OF TABLE
SZX                  SENDING OUT START ADDRESS
LIX                  READ IN X DECODE FROM TABLE
JIP OKX
JMP ERROR            See next page

OKX: SAR (INDEX)     STORE DECODE OF X IN ONE
                     OF THE MACHINE REGISTERS
LAM Y              }
ALA STRTAB         }
SZX                } USING TABLE TO FIND
                   } DECODE OF Y
LIX                }
JIP OKY
JMP ERROR            See next page

OKY: LSA           } SETTING Y DECODE TO
     LSA           } CORRECT POSITION
     BAD (INDEX)     BINARY ADDING Y DECODE
                     AND X DECODE
     ALA OPRTAB      ADDING KEYBOARD DECODE TO
     SZX             START OF OPERATING TABLE
     LIX             READING FROM OPERATING TABLE
                     (COULD BE START ADDRESS OF
                     MULTIPLY ROUTINE, OR ASCII
                     CODE OF CHARACTER ETC.)
```

Contained in Table	X or Y
0	0001
1	0010
-1	0011
2	0100
-1	0101
-1	0110
-1	0111
3	1000
-1	1001
1	1
1	1
1	1
1	1

Figure 8.1

```
ERROR:                  SEVERAL ALTERNATIVES ARE AVAILABLE
                        THE MACHINE HAS DETECTED MORE THAN
                        ONE KEY DEPRESSED AND YOU MAY WISH
                        TO SIGNAL THE ERROR OR TO IGNORE
                        THE KEYBOARD COMPLETELY.
OPRTAB:                 ADDRESS OF ROUTINE IF KEY Ø WAS PRESSED
                        ADDRESS OF ROUTINE IF KEY 1 WAS PRESSED
```

Note: OPRTAB is a list of addresses which the program starting at OKY uses to convert a key depression into a call to a particular program, for example, on a calculator the + - x ÷ keys might be pressed and the addresses in OPRTAB would be a list of the addresses of the ADD, SUBTRACT, MULTIPLY and DIVIDE routines.

Figure 8.1 (continued)

9.0 Standard Routines

This paragraph contains a variety of standarised routines which can be very useful for manipulating data stored in the 48 RAM registers on the LP8000 processor. The routines deal mainly with BCD numbers and it is assumed that two BCD digits are stored in each 8 bit register.

The main application areas for these routines are

(i) any application requiring BCD arithmetic

(ii) any application requiring bytes of data to be moved around between CPU registers or to and from main memory

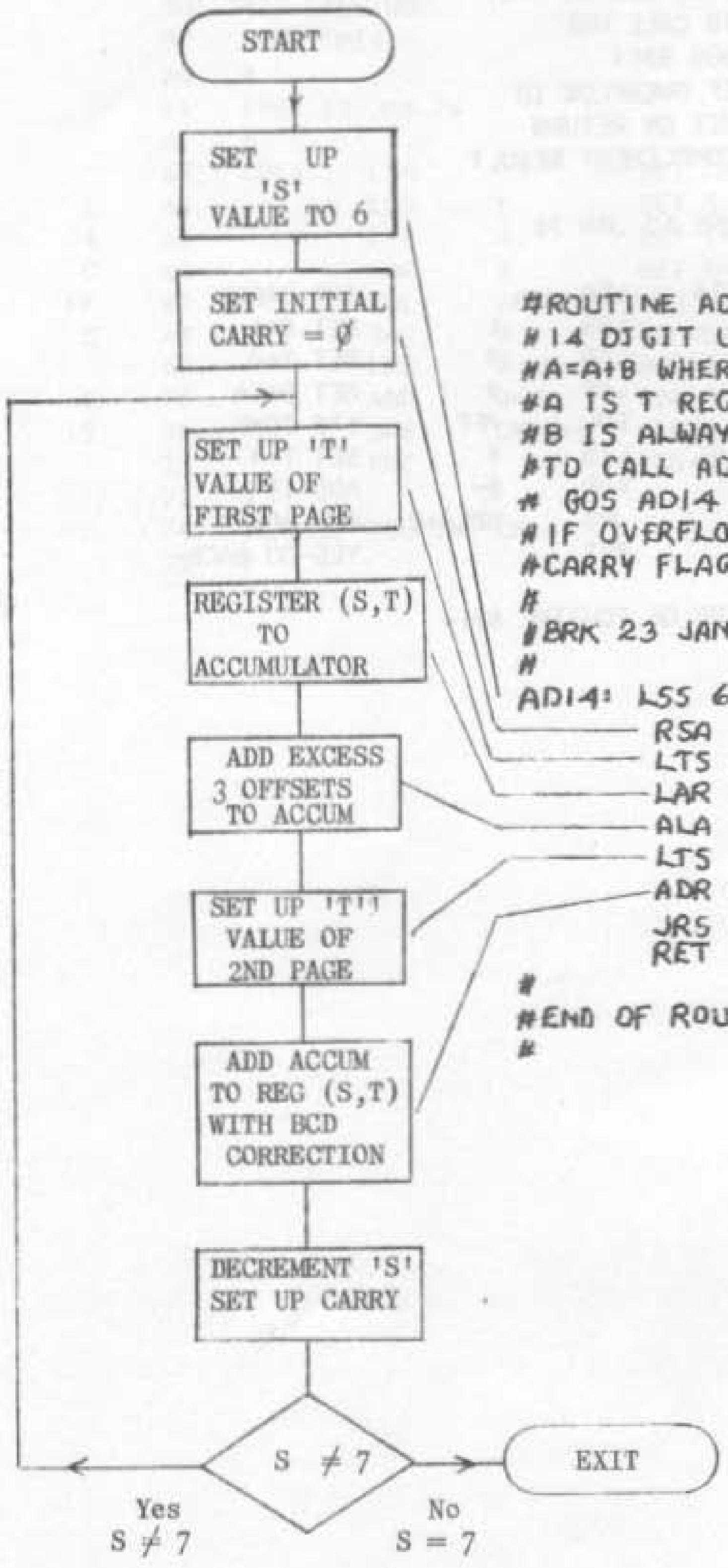
(iii) any application requiring 4 bit data manipulation, e.g. telephone numbers, security, codes, data acquision etc.

These routines are provided to serve as a source of ideas which can be adapted to suit the particular user application.

ROUTINE INDEX

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE    1
ADR  CODE   ARG  ST-NR           STATEMENT

                   1   *ROUTINE AD14
                   2   *14 DIGIT UNSIGNED ADD
                   3   *A=A+B WHERE
                   4   *A  IS T REG=2,3,4 OR 5
                   5   *B IS ALWAYS T=0
                   6   *TO CALL AD14
                   7   * GOS AD14
                   8   *IF OVERFLOW OCCURS
                   9   *CARRY FLAG IS SET ON RETURN
                  10   *
                  11   *BRK 15 JAN 76
                  12   *
000     2E     6  13   AD14    LSS    6        SET UP 'S'
001     1D        14           RSA             SET CARRY=0
002     3B        15           LTS    0        SET UP 'T'
003     8C     C  16           LAR    @        GET 'A' DIGITS
004   0E66    66  17           ALA    /66      ADD HEX 66
006     3B     3  18           LTS    3        SET UP 'B'ADR
007     ED     D  19           ADR    @-       BCD ADD,DEC 'S
008   6002     2  20           JRS    AD14+2   S NOT 7 ?
00A     00        21           RET             END OF AD14
                  22   *
                  23   *END OF ROUTINE AD14
                  24   *
```

ROUTINE TO ADD TWO 14 DIGIT BCD NUMBERS


START
SET UP 'S' VALUE TO 6
SET INITIAL CARRY = Ø
SET UP 'T' VALUE OF FIRST PAGE
REGISTER (S,T) TO ACCUMULATOR
ADD EXCESS 3 OFFSETS TO ACCUM
SET UP 'T'' VALUE OF 2ND PAGE
ADD ACCUM TO REG (S,T) WITH BCD CORRECTION
DECREMENT 'S' SET UP CARRY
S ≠ 7
EXIT
Yes
S ≠ 7
No
S = 7

```
#ROUTINE AD14
#14 DIGIT UNSIGNED ADD
#A=A+B WHERE
#A IS T REG=2,3,4 OR 5
#B IS ALWAYS T=Ø
#TO CALL AD14
#  GOS AD14
#IF OVERFLOW OCCURS
#CARRY FLAG IS SET ON RETURN
#
#BRK 23 JAN 75
#
AD14: LSS 6 SET UP 'S'
      RSA , SET CARRY=Ø
      LTS Ø SET UP 'T'
      LAR @ GET 'A' DIGITS
      ALA /66 ADD HEX 66
      LTS 3 SET UP 'B'ADR
      ADR @- BCD ADD,,DEC 'S'
      JRS AD14+2 'S' NOT 7
      RET , END OF AD14
#
#END OF ROUTINE AD14
#
```

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE     2
ADR  CODE   ARG  ST-NR          STATEMENT

                  26    $ROUTINE SB14
                  27    $SUBTRACTS TWO 14 DIGIT
                  28    $NUMBERS
                  29    $A=A-B WHERE
                  30    $A IS T REG 2,3,4,5
                  31    $B IS ALWAYS T=0
                  32    $TO CALL USE
                  33    $GOS SB14
                  34    $IF OVERFLOW IS
                  35    $SET ON RETURN
                  36    $COMPLEMENT RESULT
                  37    $
                  38    $BRK 15 JAN 76
                  39    $
00B    1C         40    SB14    LSA                 SET CARRY
00C    29     1   41            LSS      1          SET S=1
00D    38         42            LTS      0          SET T=0
00E    8C     C   43            LAR      @          GET DATA
00F  0CFF    FF   44            EOL      /FF        1'S COMP
011    3C     4   45            LTS      4          SET T=4
012    ED     D   46            ADR      @-         ADD IT
013  600D     D   47            JRS      SB14+2     ALL DONE?
015    00         48            RET                 YES-GO BACK
                  49    $
                  50    $END OF ROUTINE SB14
                  51    $
```

```
ADR  CODE   ARG  ST-NR           STATEMENT

                   53  *ROUTINE CM14
                   54  *COMPLEMENTS A 14
                   55  *DIGIT NUMBER
                   56  *A=COMP(A)
                   57  *A IS TREG=2,3,4,5,0
                   58  *TO CALL USE
                   59  *GOS CM14
                   60  *
                   61  *BRK 15 JAN 76
                   62  *
016    1C          63  CM14   LSA               SET CARRY
017    29     1    64         LSS   1           SET S=1
018    3C     4    65         LTS   4           SET T=4
019    8C     C    66         LAR   @           GET DATA
01A  0CFF    FF    67         EOL   /FF         1'S COMP
01C    9C     C    68         SAR   @           STORE BACK
01D    F0          69         LAS   0           ACC=0
01E    ED     D    70         ADR   @-          ADD 0
01F  6019    19    71         JRS   CM14+3      ALL DONE?
021    00          72         RET               YES-GO BACK
                   73  *
                   74  *END OF ROUTINE CM14
                   75  *
```

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE    4
ADR  CODE   ARG  ST-NR        STATEMENT

                  77   *ROUTINE ZR16
                  78   *16 DIGIT CLEAR ROUTINE
                  79   *CLEARS 8 WORDS OF RAM
                  80   *ADDRESSED VIA 'S' AND 'T'
                  81   *TO CALL ZR16
                  82   *T CAN BE 0 TO 6
                  83   *
                  84   *BRK 15 JAN 76
                  85   *
022    2F     7   86   ZR16    LSS     7        SET UP 'S'
023    F0         87           LAS     0        ACC=0
024    9D     D   88           SAR     @-       CLEAR @ ST
025  6024    24   89           JRS     ZR16+2   LOOP TILL S=7
027    00         90           RET              ALL DONE
                  91   *
                  92   *END OF ROUTINE ZR16
                  93   *
```

```
ADR  CODE   ARG  ST-NR          STATEMENT

                    95   *ROUTINE SL14
                    96   *14 DIGIT SHIFT
                    97   *LEFT ONE DIGIT
                    98   *
                    99   *STORE T VALUE BEFORE ENTRY
                   100   *T CAN BE 0-5
                   101   *BEWARE T=1
                   102   *
                   103   *NO TEMP STORE USED
                   104   *
                   105   *BRK 15 JAN 76
                   106   *
028   28           107   SL14     LSS     0         *S=0
029   8C       C   108            LAR     @         *GET WORD
02A   1E           109            LSN               *SHIFT + BITS L
02B   9E       E   110            SAR     @+        *STORE IN NEXT
02C  602F     2F   111            JRS     *+3       *ALL DONE?
02E   00           112            RET               *END OF ROUTINE
02F   8D       D   113            LAR     @-        *GET WORD,DEC S
030   1F           114            RSN               *SHIFT 4 BITS R
031   CC       C   115            EOR     @         *COMBINE RESULT
032   9E       E   116            SAR     @+        *STORE ACC
033  4029     29   117            JMP     SL14+1
                   118   *
                   119   *END OF SL14
                   120   *
```

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE    6
ADR  CODE   ARG  ST-NR           STATEMENT

                  122   $ROUTINE SR14
                  123   $14 DIGIT SHIFT
                  124   $RIGHT ONE DIGIT
                  125   $
                  126   $STORE T VALUE BEFORE ENTRY
                  127   $T CAN BE 0-5
                  128   $
                  129   $NO TEMP STORE USED
                  130   $
                  131   $BRK 15 JAN 76
035    2E      6  132   SR14    LSS     6         *SET UP S
036    8C      C  133           LAR     @         *GET WORD
037    1F         134           RSN               *SHIFT 4 BITS R
038    9D      D  135           SAR     @-        *STORE,DEC'S'
039  603C     3C  136           JRS     *+3       *ALL DONE?
03B    00         137           RET               *ALL DONE
03C    8E      E  138           LAR     @+        *GET WORD
03D    1E         139           LSN               *SHIFT 4 BITS L
03E    CC      C  140           EOR     @         *COMBINE
03F    9D      D  141           SAR     @-        *STORE BACK
040  4036     36  142           JMP     SR14+1    *NEXT WORD
                  143   $
                  144   $END OF SR14
                  145   $
```

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE    7
ADR  CODE   ARG  ST-NR          STATEMENT

                  147   ♦ROUTINE CR14
                  148   ♦CIRCULAR RIGHT SHIFT
                  149   ♦ONE BCD DIGIT
                  150   ♦
                  151   ♦STORE 'T' VALUE BEFORE ENTRY
                  152   ♦'T' CAN BE 0-5
                  153   ♦BEWARE T=1
                  154   ♦
                  155   ♦USES TEMP REGISTER 10
                  156   ♦
                  157   ♦BRK 15 JAN 76
                  158   ♦
042    2E     6   159   CR14    LSS     6       *SET UP 'S'
043    8C     C   160           LAR     @       *GET LEAST
044    1E         161           LSN             *SIG DIGIT
045    9A     A   162           SAR     10      *STORE TEMP
046    8C     C   163           LAR     @       *STANDARD
047    1F         164           RSN             *RIGHT
048    9C     C   165           SAR     @       *SHIFT
049  6050    50   166           JRS     *+7     *ROUTINE
04B    28         167           LSS     0       **CIRCULAR
04C    8A     A   168           LAR     10      *SHIFT
04D    CC     C   169           EOR     @       *COMBINE
04E    9C     C   170           SAR     @       *STORE
04F    00         171           RET             *END CR14
050    8E     E   172           LAR     @+      *REST OF
051    1E         173           LSN             *STANDARD
052    CC     C   174           EOR     @       *SHIFT
053    9D     D   175           SAR     @-      *ROUTINE
054  4046    46   176           JMP     CR14+4
                  177   ♦
                  178   ♦END OF CR14
                  179   ♦
```

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE   8
ADR  CODE   ARG  ST-NR              STATEMENT

                  181   *ROUTINE CL14
                  182   *CIRCULAR LEFT SHIFT
                  183   *ONE BCD DIGIT
                  184   *
                  185   *STORE 'T' VALUE BEFORE ENTRY
                  186   *'T' CAN BE 0-5
                  187   *BUT BEWARE T=1
                  188   *
                  189   *
                  190   *USES TEMP REGISTER 10
                  191   *
                  192   *BRK 15 JAN 76
                  193   *
056   28          194   CL14    LSS    0       *GET MOST
057   8C      C   195           LAR    @       *SIG DIGIT
058   1F          196           RSN            *STORE IT
059   9A      A   197           SAR    10      *IN TEMP
05A   8C      C   198           LAR    @       *NOW DO
05B   1E          199           LSN            *STANDARD
05C   9E      E   200           SAR    @+      *LEFT
05D  6064    64   201           JRS    *+7     *SHIFT
05F   2E      6   202           LSS    6       *NOW PUT
060   8A      A   203           LAR    10      *MSD
061   CC      C   204           EOR    @       *INTO THE
062   9C      C   205           SAR    @       *LSD
063   00          206           RET            *END CL14
064   8D      D   207           LAR    @-      *THIS IS
065   1F          208           RSN            *REST OF
066   CC      C   209           EOR    @       *SHIFT
067   9E      E   210           SAR    @+      *ROUTINE
068  405A    5A   211           JMP    CL14+4
                  212   *
                  213   *END OF CL14
                  214   *
```

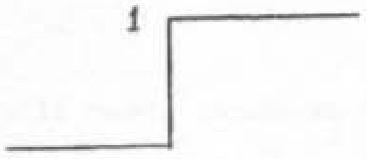
```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING   15/ 1/76 PAGE   9
ADR  CODE   ARG  ST-NR          STATEMENT

                  216   ‡ROUTINE TR16
                  217   ‡16 DIGIT TRANSFER
                  218   ‡FROM ONE 'T' REGISTER
                  219   ‡TO 'T'=0 REGISTER
                  220   ‡'T' VALUE IS IN 'V' REGISTER
                  221   ‡CALL WITH
                  222   ‡GOS TR16
                  223   ‡'T' CAN BE 1 TO 5
                  224   ‡BEWARE 'T'=1
                  225   ‡
                  226   ‡BRK 15 JAN 76
                  227   ‡
06A    2F      7  228   TR16     LSS     7       SET UP 'S'
06B    08         229            LAV             GET V
06C    01         230            SAT             STORE IN 'T'
06D    8C      C  231            LAR     @       GET NUMBER
06E    38         232            LTS     0       SET 'T'=0
06F    9D      D  233            SAR     @-      STORE NUMBER
070  606B     6B  234            JRS     TR16+1  ALL DONE?
072    00         235            RET             FINISHED
                  236   ‡
                  237   ‡END OF ROUTINE TR16
                  238   ‡
```

```
GIM LTD ALPS CROSS-ASSEMBLER   INSTRUCTION LISTING  15/ 1/76 PAGE  10
ADR  CODE    ARG  ST-NR          STATEMENT
                   240    $ROUTINE PT16
                   241    $16 DIGIT TRANSFER
                   242    $FROM ANY 'T' REGISTER
                   243    $TO ANY 'T' REGISTER
                   244    $
                   245    $'T' VALUES ARE IN
                   246    $V REG BITS 1,2,3-5,6,7
                   247    $W REG USED FOR TEMP STORE
                   248    $CALL WITH
                   249    $GOS PT16
                   250    $'T' CAN BE 0 TO 5
                   251    $BEWARE 'T'=1
                   252    $TRANSFER IS FROM V123 TO
                   253    $REGS TO V567 T REGS
                   254    $
                   255    $BRK 15 JAN 76
                   256    $
073    2F      7   257    PT16    LSS     7        SET UP 'S'
074    08          258            LAV              GET 'T' S
075    01          259            SAT              SET UP 'T1'
076    8C      C   260            LAR     @        GET DATA
077    19          261            SAW              TEMP STORE
078    08          262            LAV              GET 'T' S
079    1F          263            RSN              SHIFT 4 BITS R
07A    01          264            SAT              SET UP 'T2'
07B    09          265            LAW              GET TEMP
07C    9D      D   266            SAR     @-       STORE DATA
07D  6074     74   267            JRS     PT16+1   ALL DONE?
07F    00          268            RET              FINISHED
                   269    $
                   270    $END OF ROUTINE  PT16
                   271    $
                   272            END

 ** NO ERRORS FLAGGED IN ABOVE ASSEMBLY **
    LOADER TAPE IN FILE:- LOTEMP2
```

10.0 Generating Output Waveforms

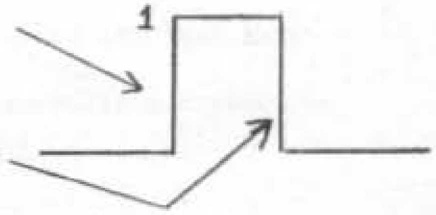
10.1 LEVEL CHANGE

```
LOAD "1" IN ACCUMULATOR              LAS 1
SEND ACCUMULATOR TO OUTPUT PORT      SAM OUTPUT 0
```



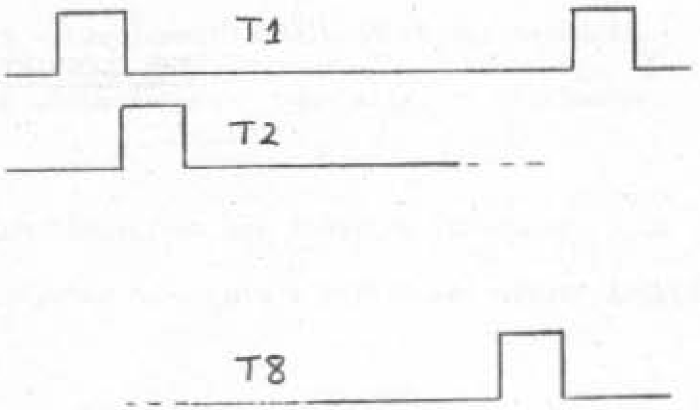
10.2 PULSE

```
LOAD (1) IN ACCUMULATOR              LAS 1
SEND ACCUMULATOR TO OUTPUT PORT      SAM OUTPUT
GO TO TIMER SUBROUTINE               GOS TIMER
LOAD (0) IN ACCUMULATOR              LAS Ø
PROCEED WITH REST OF PROGRAM         SAM OUTPUT 0
```



10.3 STROBE SEQUENCE(Display, Keyboard, etc.)

```
START:          LAS 1
STROBE:         SAR Ø
                SAM OUTPUT
                GOS TIMER
                LAR Ø
                LSA
                JIZ START
                JMP STROBE
TIMER:          LAL TIME
                SAR 1
LOOP:           DEC 1
                JNZ LOOP
                RETURN
```



Note: 'TIME' sets Multiplex Interval

SECTION E

THE GLOSSARY

GLOSSARY

ABSOLUTE	Numerical addresses and data that will remain unchanged during a loading process. A program that contains such numbers.
ABSOLUTE LOADER	A loader which will load into the computer memory a file containing absolute addresses and absolute data (as opposed to a file containing symbolic addresses or data - see RELOCATING LOADER).
ACCUMULATOR	A register in which numbers are totalled, manipulated, or temporarily stored for transfers to and from memory or external devices.
ADD	Binary (two's - complement) addition of two numbers involving the accumulator. Generally, an arithmetic operation.
ADDRESS	A number which identifies one location in memory. To direct the computer to access a particular memory location or register.
ALPHANUMERIC	The character set that contains only letters (upper and lower case) and numerals.
AND	A logical operation in which the result is true if all inputs are true and false if at least one input is false. In connection with the processor accumulator - a bit by bit logical operation in which each bit is set if both inputs are set and cleared if either or both inputs are clear.
ANSI	Abbreviation for American National Standards Institute.
ANSI FORTRAN IV	The standard subset of all FORTRAN languages.
ARGUMENT	That part of a machine code instruction which is operated on or used by the processor in executing the operation designated by the operation code (see OPERATION CODE).

ARRAY/

ARRAY	A list of elements (one dimensional array), a set of lists of elements (two dimensional), a set of sets of lists of elements (three dimensional) etc. Elements are identified in row (one dimension), by row and column (two dimensional) and by row, column and plane (three dimensional).
ASCII	Abbreviation for American Standard Code for Information Interchange. A character set and 8 bit code representation.
ASSEMBLE	To translate from a symbolic program to a binary (machine-code) program by substituting binary operation codes for symbolic operation codes (mnemonics) and absolute or relocatable addresses for symbolic addresses.
ASSEMBLER	A computer program which assembles a symbolic source program to binary machine language.
ASSEMBLER DIRECTIVE	A statement included in a symbolic source program. When the program is assembled the directive is not translated into machine language but it instructs the Assembler to perform some particular operation (sometimes called pseudo operations or instructions).
ASSEMBLY LANGUAGE	The source language used as input to an Assembler and translated by it into machine language.
AUTODECREMENT	A processor facility whereby whenever an indirect access is made the value of the pointer is decreased by one after the access (postdecrement) or before the access (predecrement).
AUTOINCREMENT	A processor facility whereby whenever an indirect access is made the value of the pointer is increased by one after the access (postincrement) or before the access (preincrement).
BINARY	The number system based on the radix two. Binary digits are restricted to 0 and 1.

BINARY CODED DECIMAL - BCD	A subset of the hexadecimal number system, in which the first ten characters are valid (decimal digits 0-9) and the upper six (hexadecimal A-F) are illegal. A binary coded decimal is a decimal number stored in four bits (weighted 8-4-2-1), with redundancy of six of the sixteen arrangements of four bits.
BCD ADD	An addition operation which accounts for the redundancy of the BCD number system and generates correct carry signals and legal BCD results.
BIT	A single digit in a binary number, or in the recorded representation of such a number. The digit can have one of two values 0 or 1.
BOOTSTRAP	A self-starting procedure which achieves a particular desired state as a result of its own action. The start of a possibly cumulative procedure.
BOOTSTRAP LOADER	A loader routine consisting of a few instructions which are sufficient to bring the rest of the loader into the memory from the input device. (Lifting itself up by its bootstraps).
CARRY	A flag or signal which becomes set or reset (true or false) as a result of an arithmetic operation generating a number greater than the base of the number system.
CENTRAL PROCESSOR	The part of a computer system which interprets and executes the instructions of a program.
CHARACTER SET	A set which includes some of all the symbols such as alphabetic letters, numerals, punctuation marks, mathematical operators etc.

CHECKSUM/

CHECKSUM	A variable given, usually at the end, of a list of items, which are to be transmitted from one device, system or program to another. The value of the variable is obtained from a (repeatable) mathematical operation upon the items in the list. The recipient can check the validity of the list by repeating the mathematical operation. The representation of the variable will depend upon the representation used in the list.
CLEAR	To set to zero, to erase the contents of something by filling it with zeros.
CODE	A system of symbols which have meaning to a computer processor, an assembler, a compiler or some other language interpreter.
COMMENT FIELD	A string of symbols, usually delimited by special symbols, contained in code,which is completely ignored by the processor or language interpreter which receives and acts upon the code. For the benefit of the user to describe and clarify the action of the program.
COMPILER	A computer program which translates the statements of a high-level language, such as FORTRAN, into machine code instructions or some intermediate form.
CONDITIONAL JUMP	An instruction which may break the sequential execution of a program, depending upon some state, flag or logical quantity. If the state is false the sequential execution is unbroken. If the state is true the instruction directs the computer to continue execution at a specified location anywhere within the memory.
CONSTANT	Numeric data used but, generally, not changed by a program.
CONTENTS	The information stored in a register or memory location.
CORE MEMORY/	

CORE MEMORY	Memory system consisting of small ferrite cores, which can be bi-directionally magnetised. Generally arranged in a three-dimensional array (a set of sets of lists) - the list represents a word and the set of sets represents the array of words. Address decoders decode the row and plane to select the addressed word.
CROSS-ASSEMBLER	An assembler written in a language other than the assembly language for running on a computer other than the computer that the assembly language program is intended for. Generally the Cross-Assembler is written in a high-level language, and is used to create the HOST-ASSEMBLER.
CURRENT LOCATION COUNTER	During assembly, a counter kept by the assembly to specify the address assigned to an instruction being assembled.
DATA ADDRESS REGISTER	A register, usually mounted with the memory (or in the same chip), which is the input to the address decoding logic of the memory system. The contents of the register are decoded to access one word of the memory. To read or write a location the Data Address Register must be loaded with the address of the location. Also called memory address register.
DATA WORD	A word, in the computer, which is interpreted by the program as containing a number, a fact (a logical quantity) or any other information.
DEBUG	To fault-find an erroneous program and prepare corrections.
DECIMAL	The number system based on the radix 10, using the digits 0-9.
DECREMENT	To decrease the contents of a register or memory word by some number, usually one.

DELIMITER/

DELIMITER	A symbol or string of symbols or other information which can be recognised by a processor or program as the start or end of some quantity.
DIRECT REGISTER MODE	When the argument of a register accesss instruction is decoded by the processor as being the address of the register to be accessed.
EXCLUSIVE-OR	A logical operation in which the output is true if one input is true and the other false (and vice versa) but false if both inputs are true or false. In connection with the processor accumulator, it is a bit by bit Exclusive-or whose output is returned to the appropriate bit of the accumulator.
EXECUTE	To perform a specific operation until its completion. As with an instruction or a program.
EXTENSION REGISTER	Address extension register. A register used to extend the addressing range of a processor. The contents of the register are appended to the most significant end of the address sent out by the processor and the whole interpreted as the memory address. The processor can change the extension by an instruction thus changing the area of addressing.
FIELD	An area of code bounded by delimiters. Its position relative to other fields or the delimiters identify the contents of the field for the processor or program.
FILE	A collection of items of information arranged in a meaningful order. Basically a paper tape or deck of punched cards.
FIRMWARE	Software that is stored permanently in Read Only Memory.
FLAG	A single binary digit which can take values of 0 or 1 (representing logical false and logical true) depending upon the condition that the flag is said to represent.

FORMAT/

FORMAT	The arrangement of bits, words, characters or strings, in a predefined way, into a statement. Fixed format - a rigorously defined arrangement, in which the absolute position within a statement is important. Free format - a loosely defined arrangement, in which the position of an item relative to other items is more meaningful than absolute position.
FORMFEED	A character of the ASCII character set whose code is recognised by lineprinters and causes them to move the paper to the top of a new page. On the teletype the form feed shifts the paper by 10 new lines.
FORTRAN	A high level language which allows programs to be written in English-like statements and algebraic equations, instead of instruction by instruction.
FORWARD REFERENCING	The facility of allowing labels and symbols to be referenced before they have been defined. Can extend to many levels (with statements referencing symbols which are equated to other symbols which have yet to be defined). Most assemblers allow only one or two levels of forward referencing.
HEXADECIMAL	The number system based on the radix 16. Hexadecimal digits include the decimal digits 0-9 and the letters A (representing 10) to F (representing 15). The hexadecimal number system fully utilizes the 16 arrangements of four bits (cf. BCD) and because of this is often used in computer systems where the word length is an integer multiple of four bits.
HOST-ASSEMBLER	An assembler written in a machine language or a machines assembly language for running on that machine to assemble a source program. The Host-Assembler would probably be assembled on a Cross Assembler.

IMAGE/

IMAGE	A direct copy of an item, a list, a register or a memory location differing only in its physical location and means of storage.
INCREMENT	To increase the contents of a register of memory location by a fixed amount, usually one.
INDIRECT ACCESS	An access of a memory location or register made by looking at the contents of another memory location or register (pointer) and using them as the address of the data to be accessed.
INDIRECT JUMP	A jump where the destination is not stated explicitly but is held in some storage area and fetched by referencing that storage area.
INDIRECT REGISTER MODE	When the argument of a register access instruction is decoded by the processor as being, not the address of a register but, an argument to a logical network for decoding to a register address - using, maybe, fixed registers or pointers.
INPUT/OUTPUT (I/O)	The transmission of information into and out of the computer system. The interface machinery for the transmission.
INTERFACE	The logic network that translates signals from the computer into some (meaningful) form for the external world and its devices.
INTER MODULE ADDRESS	The address of a location in memory expressed as a module number. This number defines the module in which the location occurs but not the location itself. (see INTRAMODULE ADDRESS)
INTRAMODULE ADDRESS	The address of a location in memory reduced to lie in the range of one module. This address specifies the position of a location within a module but not which module. Used with the INTERMODULE ADDRESS to generate the full address.

JUMP/

JUMP	An instruction which breaks the sequential execution of a program and causes the computer to continue execution at another specified location anywhere within the memory but within the program.
K	One thousand and twenty four. Thus 3K is 3072.
LABEL	Symbolic string of characters (usually alphanumeric) used instead of absolute addresses in computer programs. The label is assigned an absolute address by an assembler or compiler.
LIBRARY	A collection of standard routines, accessible by the user.
LITERAL	A constant or a self-defined symbol.
LOADER	A program which will read an input file, decode it, and set up in a computer's memory an image of it. The loader is used to load other programs into the computer, in order to execute them.
LOCATION	A group of storage elements in a computer memory, identified by a numerical address and able to store one computer word.
MACHINE CODE	The strings of binary digits 0's and 1's that the processor can interpret and execute. More rigorously, only those strings of bits that represent legal instructions.
MACRO-FACILITY	A facility of an assembler which allows each occurrence of a repeated piece of code to be replaced, in the source program, by a symbolic name - the macro name. The assembler will insert the appropriate machine code at each occurrence of the macroname.
MEMORY	A collection of storage elements organised into groups (locations), each of which can be addressed by the location number. Each location can hold one computer word. Each storage element can hold one binary digit.

MEMORY MODULE/

MEMORY MODULE	An area of memory, which is a complete entity seperate from all other areas. A segment of memory with a defined number of locations, which is usually the largest number of locations addressible by the computer word.
MICROPROGRAMMING	A method of realising logical control functions by storing the required states of signals as sequences of words in a memory.
MNEMONIC	Symbolic name for an instruction. It is recognised by the assembler and converted to the appropriate machine code.
MODIFIED HEXADECIMAL	Based on the radix 16, modified hexadecimal is a variant of the standard hexadecimal. The hexadecimal digits A-F (representing 10 to 15) are replaced by the letters J-O. In ASCII code the bottom four bits of the modified hexadecimal are identical to the fourbit coding of the hexadecimal number. Thus it is only necessary to strip off bit 5 upwards to generate the correct four bitcode.
MODULE	A conceptual building - block for a computer system. In memory, a module is the largest block of memory that can be addressed by the computer word. Thus with an eight-bit word a module is 256 locations long.
MONITOR	A supervisory program maintaining the operating environment of the computer.
NULL OPERATOR	A zero or blank operator, used where no operator is needed but where to omit one could cause confusion to a program or user.
OCTAL	The number system based on the radix 8. Legal octal digits are 0-7.
OPERAND	The item of information that is manipulated or used in an instruction. The address or symbolic name in an assembly language statement.

OPERATION CODE/

OPERATION CODE	That part of a machine-code instruction that specifies the operation to be performed by the instruction.
OPERATOR	In an assembly language statement it is the symbolic name for an instruction (the mnemonic). In algebra and highlevel languages it is a symbol which represents a particular mathematical function.
OR	A logical operation in which the output is false if neither of the two inputs is true, and true otherwise. In connection with the accumulator - a bit by bit logical operation in which each bit is cleared if neither of the corresponding bits of the inputs are set. If both or just one of the input bits is set the output bit is set also.
ORIGIN	The start of a program code. Not necessarily the start of a program but the lowest numbered location occupied by the program code. At assembly time the origin can be reset and the definition becomes:- the lowest numbered location occupied by the part of the program currently being assembled.
PACKED	When more than one piece of information is held in one unit of storage (word), e.g. two four bit BCD numbers can be packed into one 8 bit word.
PAGE	The largest area of memory that can be addressed by the computers addressing scheme. It is not necessarily the same size as a module because of Extension Registers.
PASS	A complete scanning of the input data presented to a program.
PLANE	Used in connection with a memory or register array to represent variations in a third dimension.

POINTER/

POINTER	A register or memory location containing the address of the register or location to be accessed.
POP	To remove the last entry from a push down stack or list.
PROGRAM	A sequence of computer instructions that will solve a particular problem or perform a certain series of operations.
PROGRAM AREA	The area of memory in which the program is held, as opposed to the data area, where information is held.
PROGRAM COUNTER	A special register which acts as a pointer to the next instruction to be executed. It points to a location in memory and is incremented after the instruction in that location has been executed. Thus sequential execution of the program is ensured.
PUNCHED CARDS and PAPER TAPE	Inexpensive media for data storage. Binary information is represented by punched holes. The data can be changed easily by replacing cards or splicing in new pieces of tape. Card are usually 80 columns long with 12 bits per card. Paper tape can be five, six or eight bit wide and is of (virtually) unlimited length.
PUSH	To add a new item to a pushdown list or stack.
PUSH DOWN LIST	A list of items of information in which the latest arrival is on top of the list and the oldest at the bottom. Usually operated in a LIFO manner (Last In First Out) so that items are put onto and removed from the top of the list. Often called a Stack.
RADIX	The base of a number system. The largest digit of a number system is one less than the radix, thus the number of symbols needed to represent the digits is the radix.
READ ONLY MEMORY/	

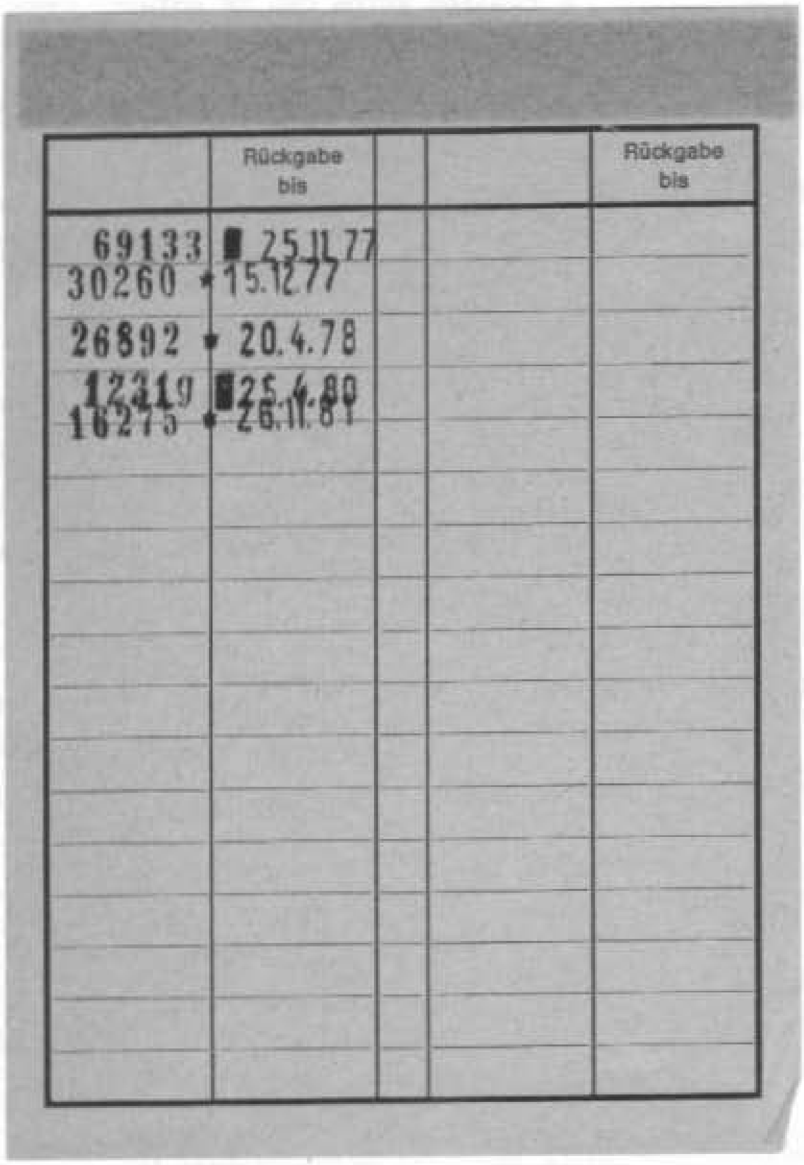
READ ONLY MEMORY	A memory in which binary data is stored by means of hardwiring. Thus a wire connected to ground may represent a zero and a wire connected to supply a one. The data stored in a read only memory cannot therefore be changed by a program.
REALTIME	The time in which a program executes.
REGISTER	A collection of a fixed number of binary storage devices, treated as complete entity rather than as grouping for the temporary storage of information. The arrangement of the connection of the devices can allow greater manipulation of the information (especially at bit level) than is allowed in a memory location.
RELOCATING LOADER	A loader which will load into the computer memory a file containing absolute, symbolic and relative addresses converting them, as necessary, to absolute addresses. The conversion procedure may be varied to account for parts of the program already loaded.
RESET	To make equal to binary zero (cf. SET). To return to the original condition, as in the case of Power On Reset which clears registers and program counters and resets status flags.
ROUTINE	A piece of self-sufficient code that can be called from a program with arguments and will perform some operation without affecting the main program, if necessary changing only the arguments.
RUNTIME	The time in which a program is executed.
SEPARATING CHARACTERS	Characters which are given special significance in certain circumstances and act as end of field marks for a program. Any character which is not itself data but appears in the data as a warning to the program processing the data.

SET/

SET	(i) To make equal to binary one (ii) A group of items with or without any special grouping relationship. A collection of pieces of information.
SHIFT	To move by a fixed amount in a recognised direction. For example, to shift left is to move to the left, to shift down is to move downwards - both by fixed amounts. In connection with the accumulator shift means to move the bit-pattern to the left or to the right by a number of bits.
SIMULATOR	A computer program which simulates under controlled conditions the operation of a system, a program or a processor. When simulating a program executing on a processor it is usual for the simulator program to be run on a larger machine using the values of variables to represent the hardward functions. The simulator will generally run slower than the system it represents.
SOFTWARE	The computer program as opposed to the computer it controls or the memory it occupies.
SOURCE PROGRAM	The program prepared by the user written in a symbolic assembly language or in a high-level language.
STACK	A list of items in which items are entered onto and removed from the top of the list. A hardware representation of such a list.
STATEMENT	A string of characters terminated by a special character such as line-feed. Treated as a group of fields by an assembler.
STORE	To put data (information) into a register or memory location, overwriting the old contents, for the purpose of retrieving the data later.
STRING	A collection of items regarded as a whole.

SUBROUTINE/

SUBROUTINE	A routine which can be entered from anywhere in a program and from which control is passed to the next sequential instruction in the main program.
SYMBOL	A meaningful string of characters.
SYMBOLIC	That which is a symbol or contains symbols.
SYSTEM ROUTINE	A standard routine or subroutine not included in, but called by a user program. Held in the MONITOR library.
TABLE	An ordered or unordered collection of items of information saved for reference purposes.
TWO'S COMPLEMENT	A representation of negative numbers so that addition and subtraction are performed similarly. The two's complement of a number is obtained by complementing and incrementing the number.
UNCONDITIONAL	Without regard for any prevailing conditions.
USER	The person who programs and operates the computer.
USER ASSIGNMENT	Of symbols, when a symbol takes a value assigned by the user rather than an assembler or compiler. A symbol assigned in the program by a direct assignment statement.
USER DEFINED	A symbol that is introduced into a program by the user, as distinct from a permanent symbol such as an instruction mnemonic.
WORD	A fixed length string of bits.
WRITE	To store data in memory (usually). To output information.



	Rückgabe bis			Rückgabe bis
69133	■ 25.11.77			
30260	• 15.12.77			
26892	• 20.4.78			
12319	■ 25.4.80			
16275	• 26.11.81			

NORTH AMERICA

UNITED STATES:
GENERAL INSTRUMENT CORPORATION MICROELECTRONICS
HEADQUARTERS—600 West John Street, Hicksville, New York 11802
Tel: 516-733-3107, TWX: 510-221-1866
NORTHEAST—Riverside Office Park, Suite 103, Riverside Road, Weston, Massachusetts 02193
Tel: 617-899-8800, TWX: 710-324-0767
SOUTHEAST—271 Schilling Circle, Hunt Valley, Maryland 21030
Tel: 301-628-2120, TWX: 710-862-9054
CENTRAL—3101 West Pratt Boulevard, Chicago, Illinois 60645
Tel: 312-338-9000, TWX: 910-221-1416
WESTERN—1100 Quail Street, Suite 114, Newport Beach, California 92660
Tel: 714-833-9400, TWX: 910-595-1730

CANADA:
GENERAL INSTRUMENT OF CANADA LTD.
61 Industry Street, Toronto M6M 4L5
Tel: 416-763-4133, TWX: 610-491-1217

ASIA

JAPAN:
GENERAL INSTRUMENT INTERNATIONAL CORP.
Fukide Building
17 Fukide-cho, Minato-ku, Tokyo 105
Tel: (03)437-0281-5, Telex: 26579

HONG KONG:
GENERAL INSTRUMENT CORPORATION MICROELECTRONICS
Room 704 Star House
3 Salisbury Road, Kowloon, Hong Kong
Tel: 3-675528, Telex: 84606

EUROPE

EUROPEAN SALES HEADQUARTERS:
GENERAL INSTRUMENT MICROELECTRONICS LTD.
57/61 Mortimer Street, London W1N 7TD
Tel: 01-636-2022, Telex: 23272

CENTRAL EUROPEAN SALES OFFICE:
GENERAL INSTRUMENT DEUTSCHLAND GmbH
(MOS Produktgruppe)
Nordendstrasse 1A, 8000 Munchen 40
Tel: (089)28.40.31, Telex: 528054

SOUTH AMERICA

BRAZIL:
GENERAL INSTRUMENT IeC Ltda
Av. Faria Lima 1794, Sao Paulo CEP 01452
Tel: 2105508

GENERAL INSTRUMENT MICROELECTRONICS