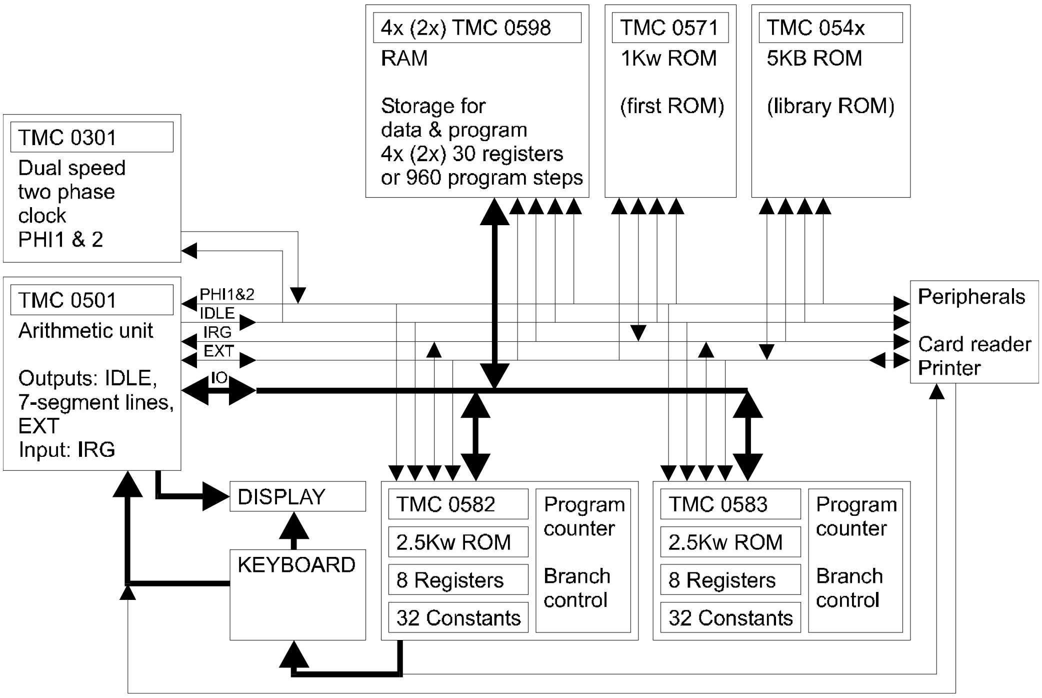
Calculators TI–58/59

HW programming guide
written by
Hynek Sladký

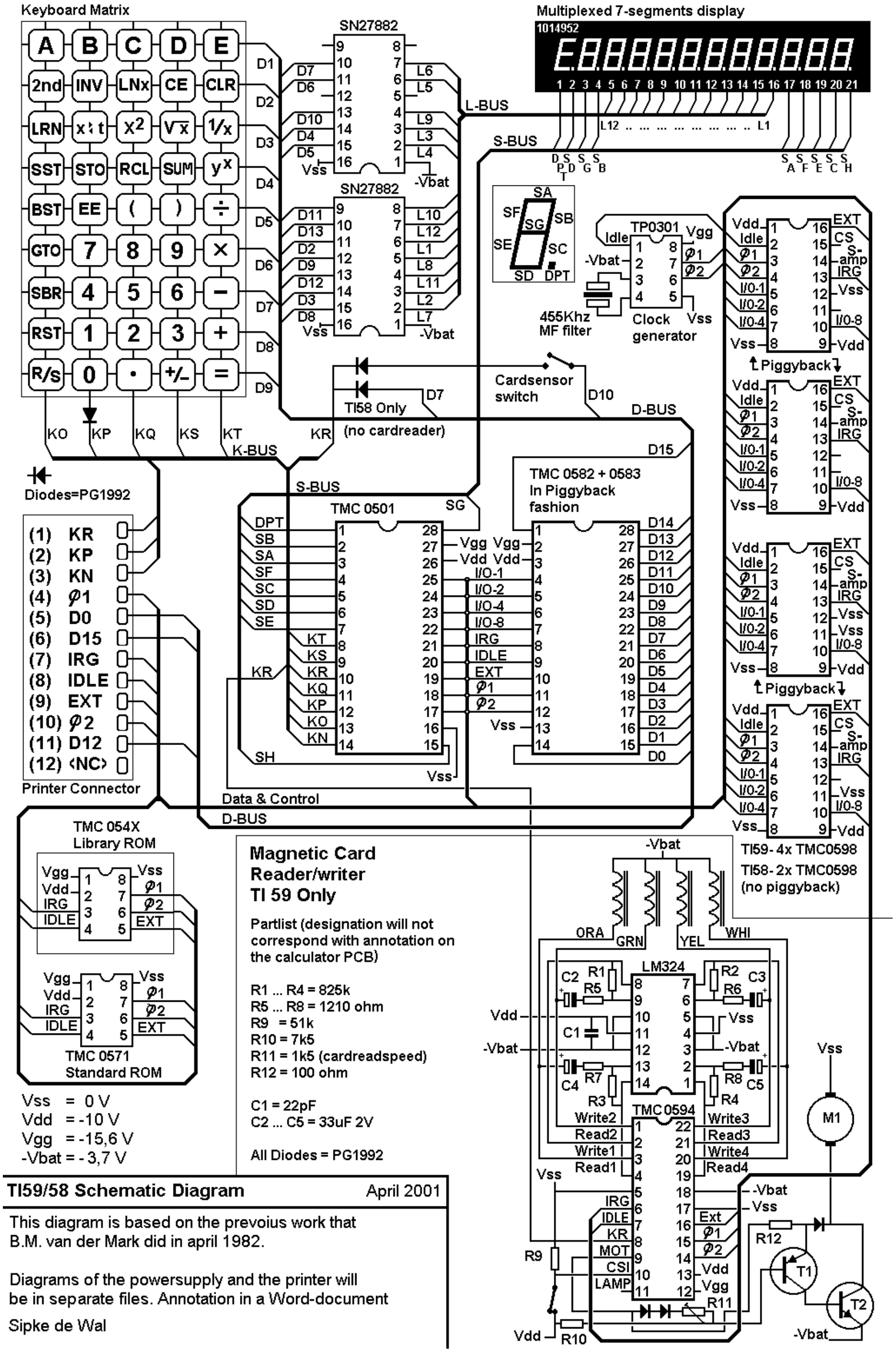
Block diagram



Calculators consist of below mentioned ICs:

TMC 0501	Control unit (CPU); contains 7-segment drivers and keyboard inputs
TMC 0582 TMC 0583	Main program memory (SCOM); contains constant memory, data registers and scanning output drivers for display and keyboard (1 double SCOM IC: 2.5Kword + 32 constants + 8 data registers)
TMC 0571	Additional program memory (13-bit „firstROM“; 1Kword; mostly printer routines)
TMC 0598	Data memory (RAM) (240 bytes/IC; 1 variable = 8 bytes/program steps; 1IC = 30 variables/240 program steps)
TMC 054x	User module (8-bit „secondROM“ for 5000 bytes)
TP 0301	Clock generator (resonator 455kHz ÷ 2 (RUN) or ÷ 8 (IDLE))

TI-59 Schematics



Magnetic Card Reader/writer TI 59 Only

Partlist (designation will not correspond with annotation on the calculator PCB)

R1 ... R4 = 825k
R5 ... R8 = 1210 ohm
R9 = 51k
R10 = 7k5
R11 = 1k5 (cardreadspeed)
R12 = 100 ohm

C1 = 22pF
C2 ... C5 = 33uF 2V

All Diodes = PG1992

Vss = 0 V
Vdd = -10 V
Vgg = -15,6 V
-Vbat = - 3,7 V

TI59/58 Schematic Diagram April 2001

This diagram is based on the prevoius work that B.M. van der Mark did in april 1982.

Diagrams of the powersupply and the printer will be in separate files. Annotation in a Word-document

Sipke de Wal

Power supply

Signal	Vmin	Vnom	Vmax	Current
Bat	3,3	3,6	3,9	160mA „0.“; 220mA „8888888888.“
Vdd	-10.5	-10.0	-9.5	40mA max
Vgg	-15.3	-15.8	-16.3	18mA max

Signals

Signal	Amplitude	10k → Vss	10k → Vampl	Ri
PHI1, 2	-15V	0.25V	0.15V	100-166
IRG	-10V	0.75V	0.6V	400-500
EXT	-10V	1.5V	0.7V	466-1000
IDLE	-10V	1.5V	0.85V	566-1000
I/O	-10V	1.2V	0.5V	333-800
Dx	-10V active 0V/330us Key scan?: -2.5V/18us			
Key	-10V open collector			
Segment	-2.5V			

Processor TMC 0501

			U			
seg DP	1	out		out	28	seg G
seg B	2	out			27	Vgg
seg A	3	out			26	Vdd
seg F	4	out	CPU	io	25	IO1
seg C	5	out	TMC	io	24	IO2
seg D	6	out	501	io	23	IO4
seg E	7	out		io	22	IO8
key T	8	in		in	21	IRG
key S	9	in		out	20	IDLE
key R	10	in		io	19	EXT
key Q	11	in		in	18	PHI2
key P	12	in		in	17	PHI1
key O	13	in			16	Vss
key N	14	in		out	15	seg H

			U			
D15	1				28	D14
Vgg	2			OVER	27	D13
Vdd	3			M-MSD	26	D12
IO1	4		SCOM		25	D11
IO2	5		TMC		24	D10
IO4	6		058x		23	D9
IO8	7				22	D8
IRG	8				21	D7
IDLE	9				20	D6
EXT	10				19	D5
PHI1	11				18	D4
PHI2	12			M-LSD	17	D3
Vss	13			E-MSD	16	D2
D0	14			E-LSD	15	D1

Signals busy and FlagB are mentioned in U.S.pat 3900722 (see figure 8b sheet 2, pins 29 and 30). They are not connected to IC pins here. Although BUSY instruction is used in firmware... This functionality is linked with KR signal... Moreover, signal KR is not scanned in firmware at all.

Display

Digit functions:

D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2
C+/-	Mantissa										
C+/-	Mantissa								+/-	Exponent	
A[13] fA[14]	A[12]	A[11]	A[10]	A[9]	A[8]	A[7]	A[6]	A[5]	A[4]	A[3]	A[2]

digit 12 = SH = FLGA => “C“ (calculate mode) is controlled with bit flgA.14 in IDLE and with all flgA bits

in RUN mode; SG+DPT function normally (i.e. minus is displayed for values 2, 3 etc.)
Flashing display = calculation error
DPT is controlled with comparator to R5 value. If R5 equals digit counter, decimal point is on for current digit.
Decoder in CPU (see USpat) contains: `0123456789AbCdEF` – values A..F weren't checked as it is not possible to get hexadecimal values in displayed digits (at least I wasn't able to do these tests).
Display is controlled by registers A and B (see table below). In RUN mode, displaying is disabled except for FLGA output. In IDLE mode, 7-segment decoder reflect registers A (value) and B (format) values and state of zero-suppression circuit. Display and keyboard are accessed from D15 down to D0 position.
Register B contains always “display mask”: normal display format is 0, minus display format is 6, space (positive sign) format is 3, to overcome zero-suppress circuit, format 9 is used. Possible display characters are summarized in table below:

A	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
B																
0		1	2	3	4	5	6	7	8	9	A	b				
1		1	2	3	4	5	6	7	8	9	A	b				
2		"	"	"	"	"	"	"	"	"	A	*	“	“	“	“
3											A	b				
4		'	'	'	'	'	'	'	'	'	A	b	'	'	'	'
5	-	°	°	°	°	°	°	°	°	°	A	8	°	°	°	°
6	-	-	-	-	-	-	-	-	-	-	A	b	-	-	-	-
7											A	b				
8	0	1	2	3	4	5	6	7	8	9	A	b				
9	0	1	2	3	4	5	6	7	8	9	A	b				
A	“	“	“	“	“	“	“	“	“	“	A	b	“	“	“	“
B											A	b				
C	'	'	'	'	'	'	'	'	'	'	A	b	'	'	'	'
D	°	°	°	°	°	°	°	°	°	°	A	b	°	°	°	°
E	-	-	-	-	-	-	-	-	-	-	A	b	-	-	-	-
F											A	b				

* this digit is combined from b and “; it looks like 8 without top horizontal line.

Number / mask examples:

Number displayed	Mask used
-12345678-77	600000009699
_12345678_77	300000009399
-1234567891_	60000000000_
_____123_45_	000008003000

Keyboard

Keyboard can be connected to inputs KN, KO, KP, KQ, KR, KS, KT.

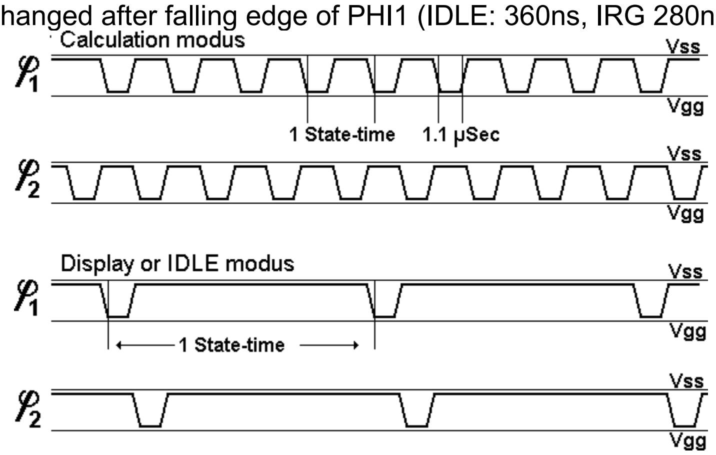
Calculator uses for keyboard inputs KO, KP, KQ, KS and KT only. KR is never used in KEY mask, it seems that KR is in reality BUSY input as this pin is used while executing TST BUSY instruction. This input is used for card reader and printer cooperation.
Input KR.D7 is used for TI-58 HW detection, KR.D10 is used for magnetic card insert detection (normally closed).
Inputs KR, KP a KN are connected to printer. (KP.D12 = PRINT, KP.D15 = TRACE, KN.D12 = ADV, KP.D0 = printer connected detection, KR = BUSY/ready)
Keyboard layout:

A' A	B' B	C' C	D' D	E' E
2nd	INV	log ln x	CP CE	CLR
Pgm LRN	P→R x ↔ t	sin x^2	cos √ x	tan 1/x
Ins SST	CMs STO	Exc RCL	Prd SUM	Ind y^x
Del BST	Eng EE	Fix (	Int)	|x| ÷
Pause GTO	x = t 7	Nop 8	Op 9	Deg ×
Lbl SBR	x ≥ t 4	∑+ 5	$\bar{x}$ 6	Rad –
St flg RST	If flg 1	D.MS 2	π 3	Grad +
Write R/S	Dsz 0	Adv .	Prt +/-	List =

The only difference in TI-58 and TI-59 keyboard is command Write as second function for R/S key on TI-59 calculator.

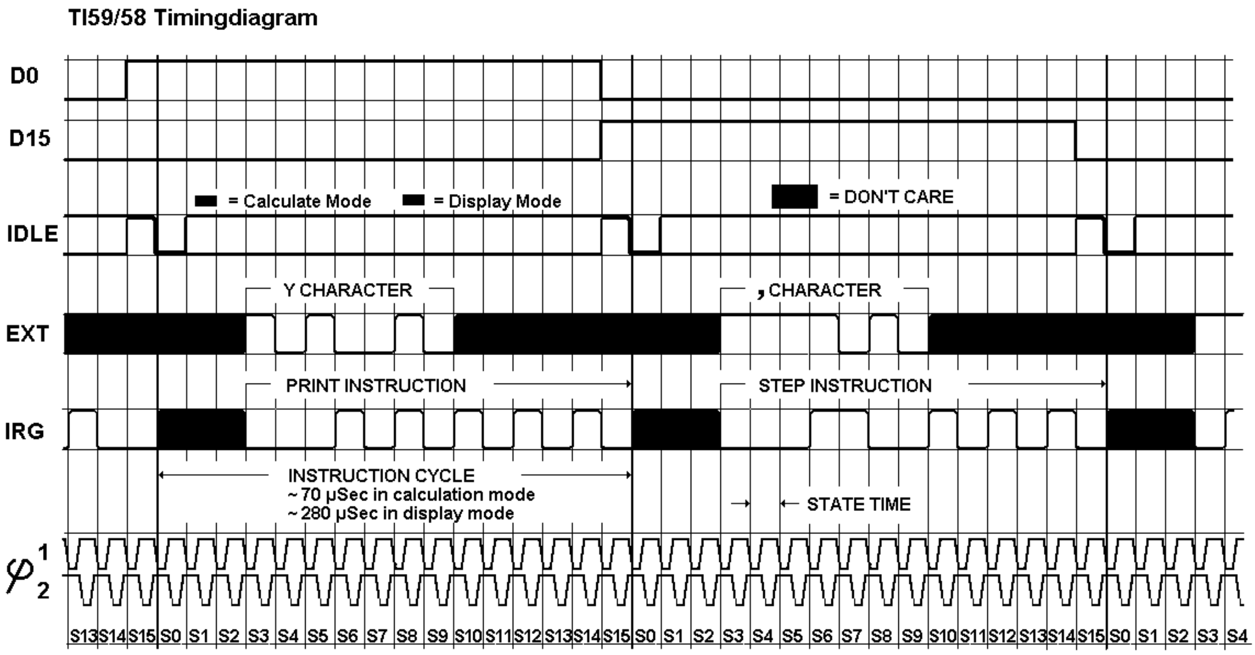
Signals PHI1 a PHI2

All ICs are clocked with signals PHI1 and PHI2. Frequency is based on resonator 455kHz. PHI1 and PHI2 are non-overlapping signals and are generated with half frequency than crystal resonator has. Active time for PHI1 and PHI2 is always about 1.1µs regardless of IDLE state. In IDLE mode, only the first from 4 cycles is generated. See waveform below.
One instruction bit period is either 4.7µs or 17.5µs. Execution speed is 14219 ips or 3555 ips. To analyze IRG+EXT, transfer speed of 56.88kB/s is required. To capture also I/O, 170.628kB/s is required. In IDLE mode, transfer rate is ¼ of RUN mode.
All output signals are changed after falling edge of PHI1 (IDLE: 360ns, IRG 280ns, EXT 560ns).



Signal IDLE

Signal IDLE is used to synchronize instruction cycle and digit count between ICs.
Instruction cycle begins with falling edge. IDLE mode is active when IDLE stays low most of the period time. RUN mode is active when IDLE is low just one instruction bit time. This RUN↔IDLE transition is detected while second instruction bit is processed, so first two instruction bits are executed with previous speed timing. All other bits use new speed timing already.
Display mode (SCOM driver output) is synchronized by transition from RUN to IDLE mode. Instruction WAIT D1 must precede SET IDLE for correct display / keyboard operation.


TI59/58 Timingdiagram
D0
D15
IDLE
EXT
IRG
■ = Calculate Mode
■ = Display Mode
■ = DON'T CARE
Y CHARACTER
, CHARACTER
PRINT INSTRUCTION
STEP INSTRUCTION
INSTRUCTION CYCLE
~70 µSec in calculation mode
~280 µSec in display mode
STATE TIME
φ 1
2
S13 S14 S15 S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S0 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12 S13 S14 S15 S0 S1 S2 S3 S4

Signal EXT

Data are sent with LSb first. First 3 bits PREG, COND, HOLD are always sent from CPU. These bits control state of instruction execution.

KR[0]	KR[15]	KR[14]	KR[13]	KR[12]	KR[11]	KR[10]	KR[9]	KR[8]	KR[7]	KR[6]	KR[5]	KR[4]	HOLD	COND	KR[1]
A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0			PREG
					CONSTANT					C53	C52	C51			0
					DATA high				DATA						0

After power-up reset, EXT signal contains value 0000 0000 0000 0xx1 for many instruction cycles to allow reliable initialization of all ICs, i.e. PREG is active and address is always 0.

PREG
Signal PREG is used to address instruction memory. This bit is automatically set after power-up reset; this bit can be controlled by KR[1] bit too. Bit KR[1] is automatically cleared after PREG is sent on EXT bus.
Instruction execution continues even PREG bit is set, so instruction used after SET KR[1] is executed as well.
Address fields:
A12-A10 chip select

A9-A7 column select
A6-A0 address
Originally, SCOM contains 1Kw instruction memory. Used double SCOM should have 2Kw instruction memory, but currently, SCOMs have 2.5Kw of instruction memory. Another 1Kw is in first-ROM.

Address range	Size	Description
0000 - 09FF	2.5Kw	TMC 0582
0A00 - 13FF	2.5Kw	TMC 0583
1400 - 17FF	1Kw	TMC 0571
1800 - 1FFF	2Kw	free

HOLD
Signal HOLD is used to wait for some external signal or to finish instruction execution (eg. WAIT instruction). This bit blocks address increment, so the same instruction is sent to CPU until HOLD bit is cleared again.

COND
Bit COND is output from ALU and TST instructions and input for branch instructions.
TST instructions can clear this bit only. Bit is set after BRA instruction is executed. If more than one BRA instruction is executed in series, COND is set after last BRA instruction.

Signal IRG

Signal IRG transports instructions from SCOM/ROM to CPU. Address counter resides in all ROM circuits and is automatically updated regarding to PREG and HOLD bits, and for BRA instructions also regarding to COND bit.
IRG format:

branch	cond	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0	dec	x	x	x
0	md	mc	mb	ma	Rd	Rc	Rb	Ra	sub	Sc	Sb	Sa			
	ALU mask				ALU operation					Destination					

ICs SCOM TMC 0582/3

Regarding U.S.pat 3900722, ICs should have 1Kw of ROM and 2 registers.
Regarding U.S.pat 4153937, double SCOM have 2.5Kw ROM, 32 constants and 8 registers.
Every SCOM contains 32 constants. Only some of them are real constants. Most of them (from address 16 up) are program tokens and work the same way as codes from Library ROM (see text 14.41 and + Table IV a Table IVa).
Constant/IO is used for ALU operations with constants from SCOM. Constant address is sent on EXT bus (KR register). Bits KR[11..8] and KR[6..4] are used as Constant address. Currently, only 2x32 constants can be addressed so one bit remains unused. ROM constants are present on I/O bus more often than ALU operations working with them; it seems that SCOMs have simplified instruction decoding for constant ROM access using less instruction bits...

SCOM Data Registers

SCOM registers are used to store internal data needed for computing.
SCOM register is accessed after Store F instruction is executed. See RCL/STO instruction with example provided.
(see U.S.pat 4153937 Fig. 19 and text 19.45...)

Digit Reg.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	IO user flags					0	0	0	RAM address			byte	Prg Src Flag	Last key		Fixed PT
									Second ROM address							
1	Hierarchy stack: mantissa													exponent		signs
2																
3																
4																
5																
6																
7																
8																
9	List data flag	0	0	0	0	0	0	0	0	Current page		New page		Security code	No. of RAMs	No. of pgm banks
10	RAM or Constant ROM program codes															
11	T register															
12	Op code1	Paren count1	OC2	PC2	OC3	PC3	OC4	PC4	OC5	PC5	OC6	PC6	OC7	PC7	OC8	PC8
	Opcode parenthesis count for hierarchy stack															
13	Page in run	0	0	0	0	0	0	RAM memory min address			RAM memory max address			No. of pending ops	Paren count	Deg Rad Grad
14	Level six					Level Five					Level Four					Cond rtn flag
	Super routine stack															
15	Super routine stack															No. sbr levels
	Level Three					Level Two					Level One					
											RAM address Const. ROM no.			Byte no.	Prog src flag	
											Second ROM address					

SCOM constant memory

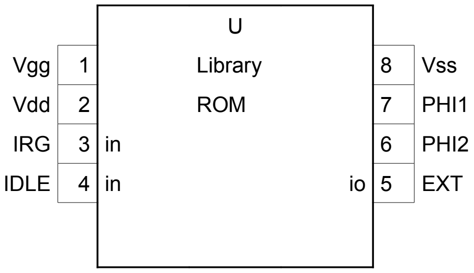
Both chips contain each 32 constants 16 digits long. Only first 16 of them are real constants. The rest are program steps. Detailed description is in Uspat 4153937 paragraphs starting with 14.32, constant data in Table V program step allocation in Table VI.

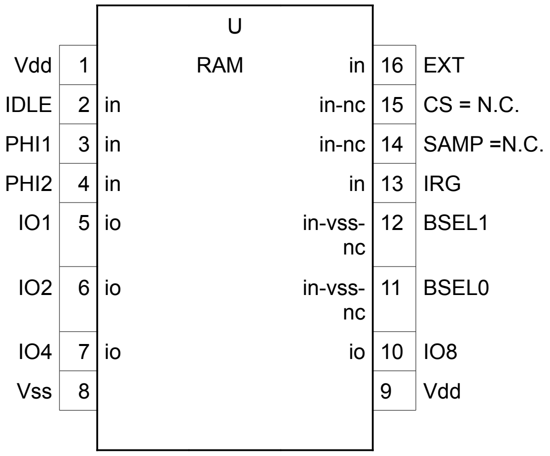
RAM memory

RAM memory is used to store program and/or data values. Generally, this memory is not used for basic computing but can be used by some extended functions like statistical calculation etc. Memory is not retained after calculator is switched off. The only exception is model TI-58C which has low-power memory chips constantly powered. TI-59 allows to store memory on magnetic cards. TI-58 doesn't have any way to save RAM contents.
TI-58 has 60 registers (two memory chips) while TI-59 has 120 registers (4 memory chips used). Because of easier register access, maximum of 100 registers is allowed to be used.
Register can hold 16-digit value or 8 bytes of program.
Memory partitioning is prepared regarding available memory found during calculator start-up routine, which checks memory cell at address 90. If this cell can hold value, memory is partitioned to default – see table below. If this cell can't hold value, TI-58 memory layout is selected with 240 program steps and 30 memory registers. TI-58 offers up to 60 registers maximum with no program space or 480 program steps with no space for registers.
Program steps are saved starting with register 0, whereas values are stored starting from last available

cell in memory for STO/RCL 00 (i.e. 59 for TI-58 or 119 for TI-59). This layout is useful when memory partitioning is changed. Saved data doesn't need to be moved.
Memory access is controlled with RAM-OP instruction, addressing and requested operation is controlled with first IO cycle whereas the second IO cycle provides data to be read or written. (Memory layout: see U.S.pat 4153937 Fig. 16)
Memory contents can be saved onto magnetic cards. Every card contains card number (i.e. which RAM page it contains) and requested memory partitioning in saved data.

	RAM a	RAM b	RAM c	RAM d
	160 program steps			100 addressable memories
	240 program steps			90 addressable memories
	320 program steps			80 addressable memories
	400 program steps			70 addressable memories
default	480 program steps			60 addressable memories
	560 program steps			50 addressable memories
	640 program steps			40 addressable memories
	720 program steps			30 addressable memories
	800 program steps			20 mem
	880 program steps			10 mem
	960 program steps			
	Card #1	Card #2	Card #3	Card #4





Library ROM

Library “Second ROM” chip is used as user changeable library module. This chip contains 5000 bytes holding library data. Library structure is described in table below (see Fig. 15 in U.S.pat 4153937 and paragraphs starting with 12.15).
ROM address pointer is part of chip. This pointer works with BCD code. It can be written through EXT bus one digit with every LOAD PC instruction. Pointer is automatically incremented after instruction FETCH is processed. Pointer value can be read out with instruction UNLOAD PC.

Address	Size	Description
0000	1	Number of pages
0001	1	Security code; Master library contains value 00
0002	2	Address of first page/program; MSB first

Address	Size	Description
0004	2	Address of second page/program
...		
N	2	Address of last page/program
N+2	2	Address of space after last page/program; this value is used to compute size of last page
N+4	1	First code from first page/program
...		
Y-1	1	Last code from last page/program
Y	1	Filled with op-code 92 = Return
...		
4999	1	

Further tables summarize program opcodes and key sequences used to enter these opcodes.

Program opcode table

Code	Function	Keys	Code	Function	Keys	Code	Function	Keys
00	0	0	34	√x	√x	68	No operation	2nd Nop
01	1	1	35	1/x	1/x	69	Operation code	2nd Op
02	2	2	36	Program Page	2nd PGM	70	Radians	2nd Rad
03	3	3	37	Polar → Rectg.	2nd P→R	71	Subroutine call	SBR
04	4	4	38	Sine	2nd sin	72	Store in indirect memory	STO 2nd Ind
05	5	5	39	Cosine	2nd cos	73	Recall indirect memory	RCL 2nd Ind
06	6	6	40	Indirect addr	2nd IND	74	Add display into indirect memory	SUM 2nd Ind
07	7	7	41	Single Step	SST	75	Minus	-
08	8	8	42	Store in mem	STO	76	Label	2nd Lbl
09	9	9	43	Recall from mem	RCL	77	Go to if x≥t	2nd x≥t
10	E'	2nd E	44	Sum into mem	SUM	78	Insert data point	2nd ∑+
11	A	A	45	y^x	y^x	79	Mean	2nd $\bar{x}$
12	B	B	46	Insert pgm code	2nd Ins	80	Grad	2nd Grad
13	C	C	47	Clear memories	2nd CMs	81	Reset	RST
14	D	D	48	Exchange display and memory	2nd EXC	82	Hierarchy address	Not directly accessible
15	E	E	49	Multiply display into memory	2nd Prod	83	Go to indirect	GTO 2nd Ind
16	A'	2nd A	50	Absolute value	2nd |x|	84	Operation code indirect	2nd Op 2nd Ind
17	B'	2nd B	51	Back step	BST	85	Plus	+
18	C'	2nd C	52	Exponent entry	EE	86	Set Flag	2nd St Flg
19	D'	2nd D	53	(	(	87	If flag set, go to	2nd If Flg
20	Clear	2nd CLR	54	)	)	88	Degrees, minutes, seconds	2nd D.MS
21	2nd	2nd	55	Divide	/	89	π	2nd π
22	Inverse Func	INV	56	Delete pgm code	2nd Del	90	List program	2nd List

Code	Function	Keys	Code	Function	Keys	Code	Function	Keys
23	LNx	LNx	57	Engineering format	2nd ENG	91	Run/Stop	R/S
24	Clear Entry	CE	58	Fixed point format	2nd Fix	92	Return	INV SBR
25	Clear	CLR	59	Integer	2nd Int	93	Decimal point	.
26	2nd	2nd 2nd	60	Degree	2nd Deg	94	Change sign	+/-
27	Inverse Func	2nd INV	61	Go To	GTO	95	Equals	=
28	log	2nd log	62	Indirect pgm page	2nd Pgm 2nd Ind	96	Write	2nd Write
29	Clear Program	2nd CP	63	Exchange indirect memory with display	2nd EXC 2nd Ind	97	Decrement register and go to when zero	2nd DSZ
30	Tangent	2nd tan	64	Multiply display into indirect memory	2nd Prod 2nd Ind	98	Advance paper	2nd Adv
31	Learn	LRN	65	Multiply	*	99	Print	2nd Print
32	Exchange display and T register	X ↔ T	66	Pause	2nd Pause			
33	x^2	x^2	67	Go to if x = t	2nd x=t			

Program codes 82 and 69 have additional parameter. Function codes are summarized in following two tables:

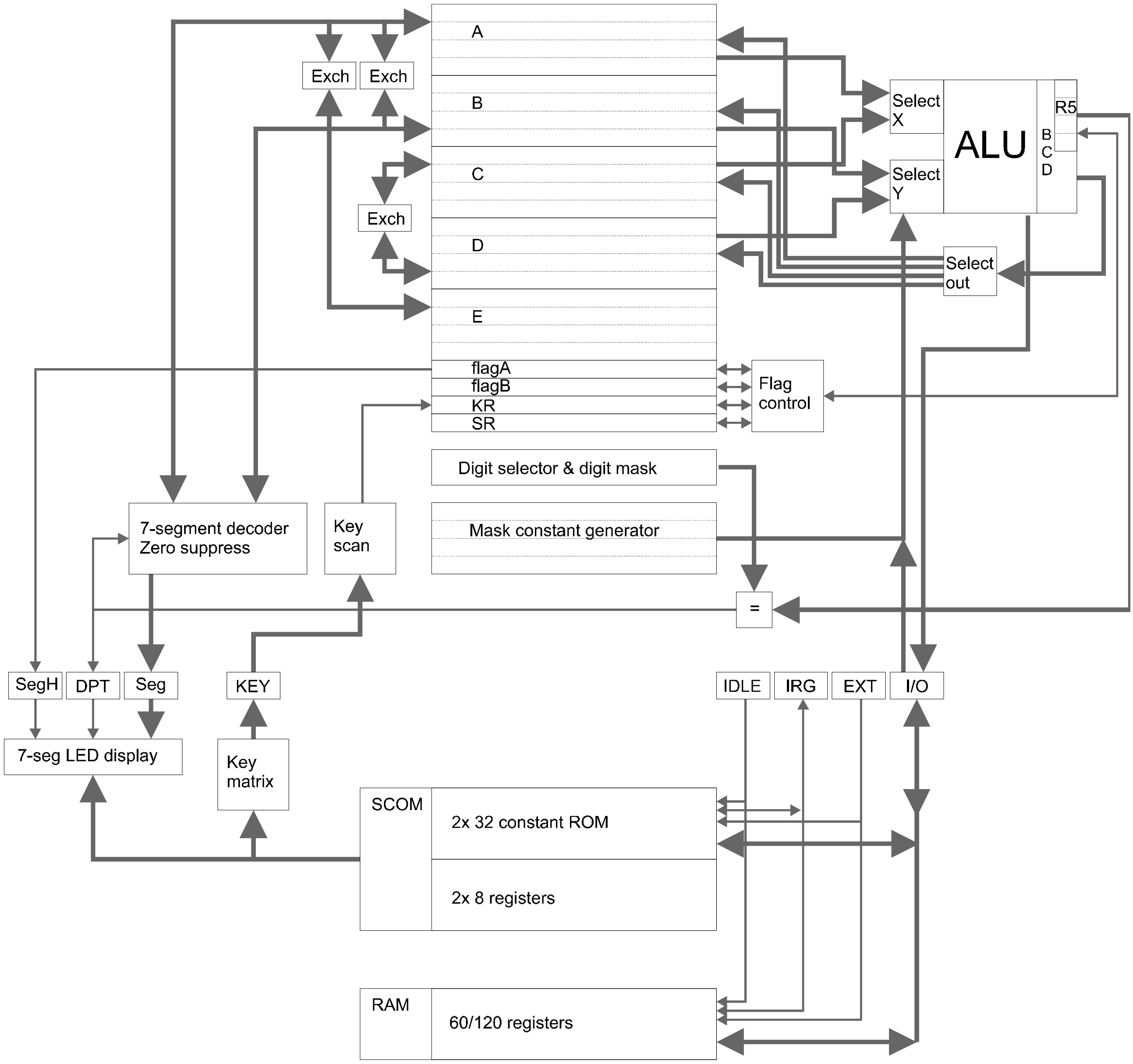
First digit	Function (82)	Second digit	Hierarchy register
0	Store	0	No operation
1	Recall	1	1
2	Conditional return; second digit is ignored	2	2
3	Sum into	3	3
4	Multiply into	4	4
5	Subtract from	5	5
6	Divide into	6	6
7	“	7	7
8	“	8	8
9	“	9	No operation

Code	Function (69)
00	Initialize for alphanumeric printing
01	Fill far left quarter of print buffer
02	Fill next to left quarter of print buffer
03	Fill next to right quarter of print buffer
04	Fill far right quarter of print buffer
05	Print the buffer as filled with OPs 01-04
06	Print display plus contents of OP 4
07	Print asterisk in column number contained in display register
08	List labels
09	Download page
10	Signum
11	Variance

Code	Function (69)
12	Slope, intercept
13	Correlation
14	y'
15	x'
16	See current partition RAM
17	Repartition RAM
18	If not error – set flag 7
19	If error – set flag 7
20 … 29	Increment memory 0 – 9
30 … 39	Decrement memory 0 – 9

CPU Programming Reference

In this chapter, CPU operating principles will be described from view of programmer. It starts with register description through instruction groups to full instruction list.
Following diagram shows principal schematics of CPU and data memories. Data paths are shown there.



CPU Registers

Registers are divided to two basic groups: digit registers (16 digits long) and bit registers (16 bits long). Register R5 is out of these two groups; it can be used for both digit and bit operations; moreover, it holds one digit result of last arithmetic operation.
One digit (4-bit registers) or bit (1-bit registers) is processed every instruction tick. Whole register (16 digits or bits) is processed in instruction cycle.

4-bit Registers

A, B are used for display in IDLE mode or as generic purpose in RUN mode.
C, D are generic purpose registers.
E is used as exchange register for values from A register only.
Number format (see U.S.pat 4153937): 16..4 = mantisa, 3..2 = exponent, 1 = signs.
R5 is used in ALU operations. This register is automatically filled with result from ALU operation on mask value digit (usually mask LSD). This register can also be used to enter 4-bit constant and to interact with flag and KR registers.

1-bit Registers

Flag A, Flag B are generic purpose flag registers. All bits of Flag A in RUN mode or bit 14 only in IDLE mode is output to SH/FLGA pin.
KR (keyboard register) is used as output for keyboard scan instruction or as input register from EXT bus. KR is used as address/data output for EXT signal as well. With PREG bit set, KR is used to change program counter programmatically.
SR (subroutine register) can be used to exchange SR and KR bits. It can be used to save KR address before KR is used as input for keyboard or EXT signal or as return address storage for “subroutine” call.

Flag instructions

Instructions used for flag 1-bit registers access.

CLR

```
CLR reg[bit]                                       flagA, flagB, KR
```

Clear requested bit in register.

```
CLR reg                                                flagA, flagB
```

Clear all bits in flag register.

```
CLR IDLE
```

Clear IDLE bit

SET

```
SET reg[bit]                                       flagA, flagB, KR
```

Set requested bit in register.
SET KR[1] sets PREG bit for EXT signal and this bit is automatically cleared immediately after execution. One more instruction is executed after PREG instruction...

```
SET IDLE
```

Set IDLE bit. To work properly, this instruction must be preceded with WAIT D1. Transition from RUN to IDLE mode is used to synchronize SCOM digit counter to CPU digit counter. If this instruction is not executed in the right digit cycle, digit counter in CPU and SCOM differ; display and keyboard results are unpredictable.

INV

```
INV reg[bit]                                           flagA, flagB
```

Invert requested bit in register.

XCH

```
XCH reg[bit],reg[bit]                                  flagA, flagB
```

Exchange bit between registers. Bit is must be the same for both registers.

```
XCH KR,SR
```

Exchange all bits between KR and SR registers. Can be used to save address prepared in KR...

MOV

```
MOV dst[bit],src[bit]                                  flagA, flagB
```

Copy bit from regS to regD. Bit number must be the same.

```
MOV KR,EXT
```

Read EXT signal and store value to KR.

```
MOV R5,reg                                        flagA, flagB, KR
```

Load R5 from KR bits 7..4 or flag bits 4..1.

```
MOV reg,R5                                        flagA, flagB, KR
```

Store R5 to KR bits 7..4 or flag bits 4..1.

TST

```
TST reg[bit]                                      flagA, flagB, KR
```

Test requested bit in register. COND is reset when tested bit is set.

```
TST BUSY
```

Test BUSY input on CPU (KR input is used for this function). COND is reset if KR input pin is set.

```
CMP reg[bit],reg[bit]                                 flagA, flagB
```

Compare bit in flag registers. Bit number must be the same for both registers. COND is reset if selected bits equal.

INC

```
INC KR
```

Increments KR register. Most significant bit is KR[0], least significant bit is KR[4]. KR value 0xFFF increments to 0x0001, 0xFFF1 increments to 0x0000. See Signal EXT description above.

Arithmetic instructions

Arithmetic instructions consist of three fields:

- mask type
- source and operation type
- destination

Mask type controls which digits are involved in arithmetic operation. Mask also holds constant which is used for some operations.

Except D0, all digits are BCD, i.e. ALU operation always makes this correction with possible carry to higher digit(s).

R5 register gets value from highlighted digit after ALU operation is executed (always the first digit processed).

List of all masks:

Name	D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
	Mantissa													Exponent		DPT
ALL	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
DPT																0
DPT1																1
DPTC																C
LLSD 1													1			
EXP														0	0	
EXP 1														0	1	
MANT	0	0	0	0	0	0	0	0	0	0	0	0	0			
MLSD 5	0	0	0	0	0	0	0	0	0	0	0	0	5			
MAEX	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
MLSD 1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
MMSD 1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
MAEX 1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

Available **operation type** is Add, Sub, Shift left, Shift right and No-operation. No-operation is used as MOV and XCH instruction. Operation type and source registers are linked – not all register combinations are possible.
ALU has two inputs X and Y. X input can be connected to A or C register or it can be zero. Y input is connected to B or D register and value is or-ed with IO digit bus. This IO digit bus is driven by mask constant and by external device. If no device is active, this bus is read as zero. It seems that IO bus is always either input or output for all ALU operations. Output is enabled explicitly selecting IO as destination. Otherwise, IO is switched to input mode, if ALU operation is executed.
All possible operation types with source registers are listed in instruction list table.
Overflow/Underflow event is signalized with COND bit cleared.
Some instructions don't use ALU for its operation – these instructions are XCH and Shift right. ALU still works and it is used to fill R5 register and possibly to drive IO bus if IO output is enabled. Instruction Shift left uses ALU before shift is made.

Destination controls where ALU result is stored. This field contains also exchange function, i.e. ALU operation is processed but not stored, and two registers exchange digits regarding mask used. ALU output can be sent to external device via IO bus.
IO bus is fed directly from ALU output, but before BCD correction takes place, so IO bus can contain hexadecimal digits on all places. All input digits are always processed with ALU — regardless of mask used! — and sent to IO bus. Carry to next digit is computed based on BCD corrected value. If such conditions occur, result value is not correct! Moreover, after this value is read back to register to be processed, BCD correction is applied and carry bits are again generated and processed. As a result, carry bits seem to be doubled for addition or even result seems to be totally wrong for subtraction.
For example: 9999999F+00000001 has correct result 00000000, but after IO operation has been processed, IO bus transfers value AAAAAAA0. After this value is read back to working register, this value becomes 11111100 due to BCD correction applied. Subtraction example looks more strange: 00000000-00000001 should be 9999999F, but real IO data is FFFFFFFF. After reading this value back to register, this value is corrected to 6666665F. (Note: these examples contain values shortened to 8 digits only.)
Arithmetic operation always changes R5 register regarding result of operation. Always result digit with position of mask constant is placed to R5 register.
List of all destinations:

Destination	Description
A	Result is written to selected register in CPU
B	
C	
D	
IO	Result is written to IO bus (used to write to SCOM register or RAM register)
AxB	Exchange contents of selected registers
CxD	
AxE	

MOV

```
MOV reg,R5                                              flagA, flagB, KR
```

Store R5 content to flag register, bits 4..1

```
MOV R5,reg                                              flagA, flagB, KR
```

Store nibble from flag register bits 4..1 to R5 register.

```
MOV R5,#const
```

Store constant nibble from instruction word to R5 register.

```
MOV reg.mask,#const                                         A, B, C, D, IO
MOV reg.mask,#-const                                        A, B, C, D, IO
```

Store mask constant to selected ALU register. This instruction uses ALU with no-operation.

ADD, SUB

```
ADD|SUB dst.mask,srcX,srcY                  A, B, C, D, IO, #0, #const
```

Arithmetic addition or subtraction. COND is reset if there is overflow or underflow on highest digit in mask. ALU input srcX can be A, C or #0. ALU input srcY can be B, D, #const or IO.
Example for constant ROM addressing and using follows:

```
0450: 0085  SET   KR[8]
      01D8  MOV   A.ALL,#-0
      0A67  MOV   R5,#6
      0A18  MOV   KR,R5
      0CC0  ADD   A.MAEX,A,const
```

SHR, SHL

```
SHR|SHL dst.mask,reg                                                   ...
SHR|SHL dst.mask,reg,#const                                            ...
```

Arithmetic digit shift to right or left. Const digit position can be or-ed before shift with constant from mask.
SHR instruction doesn't go through ALU.
SHL instruction uses ALU before shift is made. ALU operation provides BCD correction for digits D1 to D15, unfortunately before shift is made. This can lead to D1 value out of BCD range: if DPT/D0 has value higher than 9 and SHL.ALL is executed, EXP LSB/D1 receives this value without BCD correction, because DPT is not BCD corrected.

Control instructions

This group of instructions can influence program flow. These instructions can delay program execution or alter program counter value. Also SET KR[1] belongs to this group as it sets PREG bit which leads to change of program counter.

KEY

```
KEY mask                                                               ...
```

This instruction has two different ways of operation. The behavior depends on bit 3 in mask value. Other bits in mask select which inputs are used for operation. Input bits are selected for scan if appropriate bit is zero.
Note that input KR is never used in TI-58. KR input is always tested with TST BUSY instruction.

Input	KT	KS	KR	KQ	MODE	KP	KO	KN
Value	6	5	4	3		2	1	0

If MODE bit is set, keyboard inputs selected with mask are scanned immediately and COND bit is cleared if any of selected inputs is active.
If MODE bit is cleared, keyboard is scanned until digit counter reaches zero or any selected keyboard input is active. HOLD bit is set all the time keyboard scanning is active. If keyboard input is active COND bit is cleared and KR register is filled with key scancode and keyboard scanning is terminated immediately. If no key is pressed, COND bit remains set. Key input codes are mentioned in previous table, keycode format is described in following table.

0	0	0	0	0	Key input	Digit count	0	0	0	0

Keycode in KR register is often used as branch address after KR[1] is set (PREG instruction).
SR register can be used before KEY instruction to save previous KR content.

WAIT

```
WAIT Dn
```

Holds program execution until specified digit cycle. Digit time must be specified by 1 higher than requested. Note that digit counter is decremented every instruction cycle.

```
WAIT BUSY
```

...unknown behavior...

BRA

```
BRA0|1 +|-offs
```

Relative conditional jump instruction is executed in program memory ICs. Program counter doesn't reside in CPU, CPU provides control bits HOLD, COND, PREG only. These bits control program counter operation and BRA operation too.
Offset can be up to 0x3FF, i.e. +/-1023. COND bit is set after executing last jump instruction in series, i.e. if more than one jump is executed (long jump needed), COND bit remains the same (reset) for all jumps in series.
There are two special BRA opcodes: BRA +1 is used to clear COND bit (opcodes 1002 and 1802); BRA 0 is used in debugger as HALT instruction (1001 and 1801).

Peripheral instructions

Peripheral instructions control peripheral behavior. CPU executes no-operation. Some instructions use or generate data on EXT signal, some instructions use data on IO bus. If IO bus is used, read access takes part of data selected by mask, whereas write access stores all digits from IO bus.

STO, RCL

Instructions are used to control SCOM register access.
SCOM write access is executed after Store F instruction is executed. Address is taken from last I/O access before STO/RCL instruction. Note that if no IO access precedes RCL or STO instruction, address of 0 is used.
It seems, that SCOM registers can be accessed from own program ROM only. STO and RCL instructions are decoded immediately after loaded from ROM array to be shifted out of SCOM.
SCOM write access example:

```
      0121  MOV   IO.ALL,C
      0A0F  STO
      0101  MOV   IO.ALL,A
```

SCOM read access example:

```
      0111  MOV   IO.ALL,B
      0A1F  RCL
      01D4  MOV   C.ALL,#0          ;read IO
```

RAM_OP

This instruction is used to control RAM access. Next instruction cycle must hold command and address for RAM access. RAM address uses digits 2 and 3 (IO[3]*10+IO[2]). If write access is requested, over-next instruction cycle IO data are written to specified RAM register. If read access is requested, over-next instruction cycle IO data are read and copied regarding used mask to destination register.

												ADDR		OP

RAM commands:

RAM command	Description
0	read RAM register
1	write RAM register
2	clear 1 RAM register
4	clear 10 successive RAM registers

RAM write access example:

```
      03D0  MOV   A.DPT,#1
      0AF8  RAM_OP
      0101  MOV   IO.ALL,A
      0101  MOV   IO.ALL,A
```

RAM read access example:

```
      02D8  MOV   A.DPT,#-0
      0AF8  RAM_OP
      0101  MOV   IO.ALL,A
      0CD3  MOV   B.MAEX,#0        ;read IO
```

IO bus is in output mode for all ALU operations with IO as output destination. Otherwise IO is input. This can be confirmed with ALU operation not using IO but with IO bus active (e.g. RAM output active).

LIB

These instructions are used to access Second ROM library chip. All data transfer is done through EXT bus.
Two instructions access library module address pointer. This access is done by one BCD nibble.

```
OUT LIB_PC
```

One nibble from EXT bus (bits 7 to 4 in KR register) is written to library address register. Before nibble is written, address is shifted to the right and most significant nibble is written.

```
IN LIB_PC
```

Least significant nibble is read out from library address register. The contents of this register is then shifted right by one nibble.

Another two instructions are used to get data out of library module chip. Byte addressed with address pointer is sent through EXT signal.

```
IN LIB
```

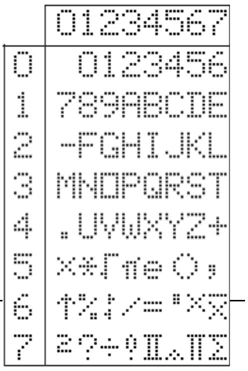
EXT signal contains whole byte of data from library ROM. After MOV KR,EXT instruction is executed, KR contains this byte in bits 11 to 4. Moreover, internal address register is incremented after data has been sent. TI-58/9 uses low nibble only.

```
IN LIB_HIGH
```

EXT signal contains high nibble only. This nibble can be read with MOV KR,EXT to bits 7 to 4 of KR register. Address register is not incremented.

PRT

Printer is controlled with TMC0251. Some aspects are described in U.S. Pat. 4020465. Communication principles are mentioned in TI-59 service manual. Printer codepage table is provided in U.S. Pat. 4153937.
This chapter provides mostly programming description – i.e. instructions used for printer control. Printer control is placed through all ROM. There is no specific place with print routines only.
Connection uses these signals: KR, KP, KN, Phi1, D0, D15, IRG, IDLE, EXT, Phi2, D12



	0	1	2	3	4	5	6	7
0		0	1	2	3	4	5	6
1	7	8	9	A	B	C	D	E
2	-	F	G	H	I	J	K	L
3	M	N	O	P	Q	R	S	T
4	.	U	V	W	X	Y	Z	+
5	×	*	√	π	e	(	)	,
6	↑	%	‡	/	=	'	X	x̄
7	²	?	÷	!	Ⅱ	▲	Π	Σ

Printer detection: D0-KP
Control buttons: ADV = D12-KN; PRINT = D12-KP; TRACE = D15-KP (this control is on/off button instead of push button; in TRACE mode, this signal is permanently connected)
Signal BUSY (KR) is connected directly from TMC0251 to CPU. It is probably valid after STEP or FEED instruction has been executed.
Printer uses special character table (see picture above).
Printer has possibility to print names 3 characters long using 1 data byte. All known function names are summarized in table below:

Code	Text														
00	___	17	_×_	23	DPT	33	x^2_	53	PRM	61	SUM	70	ERR	76	HLT
11	_=_	1A	x√Y	26	CE_	36	1/x	54	_%_	66	STO	71	_(_	78	STP
12	_-_	1B	Y^x_	27	+/-	3C	√x_	56	COS	67	_π_	72	_)_	7A	GTO
13	_+_	21	CLR	2D	EE_	3D	X↔Y	57	SIN	68	RCL	73	LRN	7C	IF_
16	_÷_	22	INV	31	e^x_	51	LNx	5D	TAN	69	∑+_	74	RUN		

Printer has 20 character buffer which is addressed from right to left. i.e. texts are entered last character first...

```
OUT PRT
```

Character is added from EXT signal (KR bits 4 to 9) to print buffer and buffer pointer is moved 1 position to the left.

```
OUT PRT_FUNC
```

Add function name (3 characters – see table above) to the print buffer and move buffer pointer accordingly.

```
PRT_CLEAR
```

Clear print buffer and initialize buffer pointer to most right position.

```
PRT_STEP
```

Fill current buffer position with blank character and move buffer pointer 1 position to the left. Also output busy signal to KR next instruction cycle to be tested with TST BUSY instruction.

```
PRT_PRINT
```

Print characters from buffer to the paper and advance paper by one print line.

```
PRT_FEED
```

Feed paper by half print line. Printer is busy (see PRT_STEP instruction) until paper is moved.
When simulating, there should be some time holding KR input when this instruction is processed — TI-58 sends PRT_FEED all the time printer or calculator button is held as soon as KR signal is released. If KR is not emulated for this instruction, PRT_FEED is sent repeatedly in fast loop producing more than expected paper feed.

CRD

Reader detection: no D7-KR connection (TI-59 mode)
Card detection: D10-KR (normally closed)
Technical details are described in U.S.pat. 4006455.
Reader chip can control HOLD and COND bits.
HOLD bit can be activated when executing CRD_READ or CRD_WRITE instruction and card reader chip needs more time to complete requested task (see Fig. 2 and 3 in U.S.pat.).
Reader chip has some basic error check built in. If there are two and more errors, COND bit is activated. COND bit is activated also in conjunction with “Card Sense Input”; this function is described as protection and it seems that it was intended to provide hardware write protection for read-only cards. I haven't found any information about this function.
COND bit is driven after CRD_OFF instruction is executed and until BRA instruction clears again this bit. It seems, that this bit is checked for read access only. CRD_OFF for write access doesn't use this

bit and immediately COND clear instruction is executed.

This group of instructions is used to control card reader. Routines used to work with card reader are placed in ROM at addressed 16B2 to 1796.

```
CRD_OFF
```

Switch reader off. If COND bit is active, it is output until BRA is executed.

```
CRD_READ
```

Switch reader on for reading. HOLD bit can be active if card reader is not ready to continue.

```
CRD_WRITE
```

Switch reader on for writing. HOLD bit can be active if card reader is not ready to continue.

```
IN CRD
```

Reader chip sends 8 bits of data read from card to EXT bus. Must be always preceded with CRDREAD instruction and followed with MOV KR,EXT instruction. Data bits are then placed to bits KR[11..4].

```
OUT CRD
```

Reader chip accepts 8 bits of data from EXT bus to be written to card. Must be always preceded with CRDWRITE instruction. Data bits are taken from KR[11..4].

Card data structure

Size	Format	Description
1	`1n`	Memory partitioning information. Holds page count (10 registers long) for program storage. Possible values are 12, 13, … 1C
1	`1n`	Data type on card. 11 is for program, 10 is for data or program/data card.
1	`1n`	Page number. Can be 10, 13, 16 and 19 for cards #1, 2, 3 and #4.
1	`n0`	Protection status. 00 if program is unprotected, 10 means program is protected against listing, single-stepping, interrupting or another debugging technique.

30 x 8	`ab cd ef gh`
	`ij kl mn op`
		Data starting with first register. `badcfehgjilkn` is mantissa, `mp` is exponent, `o` holds sign bits (digit 0 in register). Program codes are stored starting with digits 1 and 0 (i.e. nibble swapped): `po nm lk ji hg fe dc ba`.

2	`0n 0n`	Check sum bytes. (twice the same value)

Examples:

`12 10 10 00` … Card #1 for 159.99 partitioning
`13 11 10 00` … Card #1 for 239.89 partitioning
`1C 11 10 00` … Card #1 for 959.00 partitioning
`16 11 13 00` … Card #2 for 479.59 partitioning
`16 10 16 00` … Card #3 for 479.59 partitioning
`16 10 19 00` … Card #3 for 479.59 partitioning

CPU instruction list

List of all known instructions:

Op-code	Instruction	Mnemonic	COND	R5	Description
`0 0000 ssss 0000`	`TEST FLAG A`	`TST FA[s]`	•		Test bit in flagA register
`0 0000 ssss 0001`	`SET FLAG A`	`SET FA[s]`			Set bit in flagA register
`0 0000 ssss 0010`	`ZERO FLAG A`	`CLR FA[s]`			Clear bit in flagA register
`0 0000 ssss 0011`	`INVERT FLAG A`	`INV FA[s]`			Invert bit in flagA register
`0 0000 ssss 0100`	`EXCH. FLAG A B`	`XCH FA[s],FB[s]`			Exchange bit between flagA and flagB registers
`0 0000 ssss 0101`	`SET FLAG KR`	`SET KR[s]`			Set bit in KR register

`0 0000 0001 0101`	`PREG`
	`SET FLAG KR[1]`
		`SET PREG`			Set KR[1] bit; this bit is then sent as PREG to EXT bus and then cleared

`0 0000 ssss 0110`	`COPY FLAG B → A`	`MOV FA[s],FB[s]`			Copy bit from flagB to flagA register

Op-code	Instruction	Mnemonic	COND	R5	Description
0 0000 oooo 0111	REG 5 → FLAG A	MOV FA,R5			Set flagA[4..1] according to R5 value
0 0000 ssss 1000	TEST FLAG B	TST FB[s]	•		Test bit in flagB register
0 0000 ssss 1001	SET FLAG B	SET FB[s]			Set bit in flagB register
0 0000 ssss 1010	ZERO FLAG B	CLR FB[s]			Clear bit in flagB register
0 0000 ssss 1011	INVERT FLAG B	INV FB[s]			Invert bit in flagB register
0 0000 ssss 1100	COMPARE FLAG A B	CMP FA[s],FB[s]	•		Compare bit from flagA and flagB registers
0 0000 ssss 1101	ZERO FLAG KR	CLR KR[s]			Clear bit in KR register
0 0000 ssss 1110	COPY FLAG A → B	MOV FB[s],FA[s]			Copy bit from flagA to flagB register
0 0000 oooo 1111	REG 5 → FLAG B	MOV R5,FB			Set flagB[4..1] according to R5 value
0 0001 rrrr sSSS	All Mask	.ALL			[0000000000000000]
0 0010 rrrr sSSS	DPT	.DPT			[_______________0]
0 0011 rrrr sSSS	DPT 1	.DPT1			[_______________1]
0 0100 rrrr sSSS	DPT C (4)	.DPTC			[_______________C]
0 0101 rrrr sSSS	LLSD 1 (9)	.LLSD1			[____________1___]
0 0110 rrrr sSSS	EXP	.EXP			[_____________00_]
0 0111 rrrr sSSS	EXP 1 (9)	.EXP1			[_____________01_]
0 1000 TSRQ 0PON	keyboard	KEY mask	•		Scan keyboard starting with current digit output until digit 0 is reached or until key press is detected. This instruction is often preceded with WAIT Dn instruction. Output is stored in KR[10..4] bits = [ccc] [rrrr], where: ccc=0..6 for KN, KO, KP, KR, KS, KT inputs; current keyboard uses rows 1, 2, 3, 5, 6 only (left to right) rrrr=0..15 for D0 to D15; current keyboard uses rows 1..9 only (top to bottom)
0 1000 TSRQ 1PON	keyboard	KEY mask	•		One keyboard row test. Masks used: FB, FD, FE, EF – all have only one input active.
0 1001 rrrr sSSS	MANT	.MANT			[0000000000000___]
0 1010 dddd 0000	WAIT D	WAIT digit			Wait until specified digit time arrives (/D)
0 1010 oooo 0001	Zero Idle	CLR IDLE			Clear IDLE bit (switch to RUN mode)
0 1010 oooo 0010	CLFA	CLR FA			Clear all flagA bits
0 1010 0011	Wait Busy	WAIT BUSY			(never used in TI-58)
0 1010 oooo 0100	INCKR	INC KR			KR[0,15..4] value increment (KR[0] is top most bit!)
0 1010 ssss 0101	TKR	TST KR[s]	•		Test bit in KR register
0 1010 ooo0 0110	COPY FLGA → R5	MOV R5,FA		•	Set R5 to flagA[4..1] value
0 1010 ooo1 0110	COPY FLGB → R5			•	Set R5 to flagB[4..1] value
0 1010 dddd 0111	Number	MOV R5,#const		•	Put number into R5
0 1010 0000 1000	KR → R5	MOV R5,KR		•	Load R5 with LSD of keyboard reg KR[7..4]
0 1010 0001 1000	R5 → KR	MOV KR,R5			Load LSD of keyboard reg KR[7..4] with R5
0 1010 0010 1000	DR8EXT	IN CRD			Output data from card reader chip to EXT (8 bits: KR[11..4])
0 1010 0011 1000	EXTDR8	OUT CRD			Write data from EXT (8 bits KR[11..4]) to card reader chip
0 1010 0100 1000	TOFF	CRD_OFF	•		Switch card reader off
0 1010 0101 1000	RDON / RDNT	CRD_READ			Switch card reader to read mode. If reader is not ready, this instruction can drive HOLD bit.
0 1010 0110 1000	LOAD	OUT PRT			Write to print buffer and decrement pointer (6 bits: KR[9..4])
0 1010 0111 1000	FUNCTION	OUT PRT_FUNC			Write function name to buffer (7-bit code: KR[10..4])
0 1010 1000 1000	CLEAR	PRT_CLEAR			Clear print buffer and reset print position @ 20
0 1010 1001 1000	STEP	PRT_STEP			Decrement print buffer pointer
0 1010 1010 1000	PRINT	PRT_PRINT			Set print buffer pointer to 0 (i.e. starts printing)
0 1010 1011 1000	PAPER ADVANCE	PRT_FEED			Paper feed
0 1010 1100 1000	WRON	CRD_WRITE			Switch card reader to write mode. If reader is not ready, this instruction can drive HOLD bit.

Op-code	Instruction	Mnemonic	CO ND	R5	Description
0 1010 1111 1000	RAM in/out	RAM_OP			RAM access instruction. RAM address and command is decoded from IO bus on next instruction cycle. RAM data is transferred (based on command: from or to RAM) in after next IO cycle.
0 1010 oooo 1001	Set Idle	SET IDLE			Set IDLE bit, i.e. switch to IDLE mode (displaying is enabled, CPU runs slow) Transition from RUN to IDLE synchronizes CPU with SCOM; for correct behavior, this instruction must bee preceded with WAIT D1
0 1010 oooo 1010	CLFB	CLR FB			Clear all flagB bits
0 1010 oooo 1011	Test Busy	TST BUSY	•		Test BUSY input signal. This signal is connected to KR input. To scan requested input, WAIT Dn must precede TST BUSY.
0 1010 0000 1100	EXT KR	MOV KR,EXT			Load keyboard reg with EXT data. If no data is currently on the EXT bus, this instruction clears KR register.
0 1010 oooo 1101	XKRSR	XCH KR,SR			Exchange SR and KR bits
0 1010 oo?? 1110	NO-OP				Instructions for peripherals
0 1010 0000 1110	FETCH	IN LIB			Load byte from SecondROM through EXT to KR[11..4] in next instruction cycle with automatic address post-increment.
0 1010 0001 1110	LOAD PC	OUT LIB_PC			Put 4-bit part of address counter to SecondROM through EXT from KR[7..4]
0 1010 0010 1110	UNLOAD PC	IN LIB_PC			Load 4-bit part of address counter from SecondROM through EXT to KR[7..4] in next instruction cycle
0 1010 0011 1110	FETCH HIGH	IN LIB_HIGH			Load 4-bit high nibble from SecondROM through EXT to KR[7..4] in next instruction cycle
0 1010 rrr0 1111	Register				SCOM register write (TI-58 uses opcode 0A0F only)
0 1010 rrr1 1111					SCOM register read (TI-58 uses opcode 0A1F only)
0 1010 0000 1111	Store F	STO F			...
0 1010 0010 1111	Store G				Not used in TI-58
0 1010 0001 1111	Recall F	RCL F			...
0 1010 0011 1111	Recall G				Not used in TI-58
0 1011 rrrr sSSS	MLSD 5	.MLSD5			[0000000000005___]
0 1100 rrrr sSSS	MAEX	.MAEX			[000000000000000_]
0 1101 rrrr sSSS	MLSD 1	.MLSD1			[0000000000001oo_]
0 1110 rrrr sSSS	MMSD 1	.MMSD1			[1000000000000oo_]
0 1111 rrrr sSSS	MAEX 1	.MAEX1			[000000000000001_]
0000 0	A+<mask>	ADD _,A,#const	•	•	
0000 1	A-<mask>	SUB _,A,#const	•	•	
0001 0	B|<mask>	OR _,B,#const	•	•	
0001 1	-(B|<mask>)	NEG _,B|#const	•	•	Nonzero mask only once (071B=SUB.EXP B,#0,B|#1)
0010 0	C+<mask>	ADD _,C,#const	•	•	
0010 1	C-<mask>	SUB _,C,#const	•	•	
0011 0	D|<mask>	OR _,D,#const	•	•	
0011 1	-(D|<mask>)	NEG _,D|#const	•	•	Never used with nonzero mask.
0100 0	Shift left A	SHL _,A[,#const]		•	Const is or-ed with register before shift... But never used with nonzero mask in TI-58...
0100 1	Shift right A	SHR _,A[,#const]		•	
0101 0	Shift left B	SHL _,B[,#const]		•	
0101 1	Shift right B	SHR _,B[,#const]		•	
0110 0	Shift left C	SHL _,C[,#const]		•	
0110 1	Shift right C	SHR _,C[,#const]		•	
0111 0	Shift left D	SHL _,D[,#const]		•	
0111 1	Shift right D	SHR _,D[,#const]		•	
1000 0	A+B	ADD _,A,B[|#const]	•	•	

Op-code	Instruction	Mnemonic	COND	R5	Description
1000 1	A-B	SUB _,A,B[|#const]	•	•	
1001 0	C+B	ADD _,C,B[|#const]	•	•	
1001 1	C-B	SUB _,C,B[|#const]	•	•	
1010 0	C+D	ADD _,C,D[|#const]	•	•	
1010 1	C-D	SUB _,C,D[|#const]	•	•	
1011 0	A+D	ADD _,A,D[|#const]	•	•	
1011 1	A-D	SUB _,A,D[|#const]	•	•	
1100 0	A+constant/io	ADD _,A,IO[|#const]	•	•	Operations with data from SCOM constant ROM... Never used with nonzero mask (all IO instructions for A and C)
1100 1	A-constant/io	SUB _,A,IO[|#const]	•	•	
1101 0	NO-OP	MOV _,#const	?	•	Load data from IO bus; used for SCOM/RAM reading Behavior if mask is not zero??? IMHO or-ing with loaded value... (Used for MOV reg,#<mask>)
1101 1	NO-OP	MOV _,#-const	•	•	Used for MOV reg,#-<mask>
1110 0	C+constant/io	ADD _,C,IO[|#const]	•	•	
1110 1	C-constant/io	SUB _,C,IO[|#const]	•	•	
1111 o	R5 → Adder	MOV _,R5[|#const]	?	•	Mask LSD Never used with Sub operation... Used once with nonzero mask (0EF3=MOV.MMSD B,R5|#1 ??)
000	Σ → A	___ A,			
001	Output I/O	___ IO,			Send result to I/O bus
010	A ↔ B	XCH A,B			ALU operation still executes but without storing result!
011	Σ → B	___ B,			
100	Σ → C	___ C,			
101	C ↔ D	XCH C,D			ALU operation still executes but without storing result!
110	Σ → D	___ D,			
111	A ↔ E	XCH A,E			ALU operation still executes but without storing result!
1 Caaa aaaa aaa0	Branch +A	BRA0 offs BRA1 offs	•		Branch if COND = bit C COND bit is set after last BRA instruction in series
1 Caaa aaaa aaa1	Branch -A		•		
x x11x 0xxx xxxx					Recall constant – see SCOM decoder But regarding to captured data it is not right... I would expect 0A.F opcodes

External debugger

Debugger used to discover many secrets of TMC-0501 and other chips in TI-58.
Based on STM32F4xx MCU running at 64MHz.
Debugger can drive EXT and IRG signals. If user program has to be run, EXT signal is driven with address and PREG bit set to force all program chips to jump to desired address. This signal is sent until HOLD bit is detected active to be sure that all chips take this new address into account. It seems that HOLD bit has higher priority than PREG bit...
Debugger uses unused ROM area from address 0x1800 up.
EXT and IRG signals are monitored to see instruction trace. Sometimes EXT signal was used to trace output data. Later, test programs used display to show test results.

Test program examples

Simple count test

This test simply displays counter. It assumes IDLE mode is selected. Incrementing speed is about 1185 loops per second.

```
1800:   01D8     MOV      A.ALL,#0
        01DB     MOV      B.ALL,#0
1802:   0D00     ADD      A.MLSD,A,#1
        0A37     MOV      R5,#3
        1805     BRA1     -2            ;1802
        1007     BRA0     -3            ;1802
```

Stopwatch

This example is more complex. It uses WAIT Dn instruction to make timing more precise (without counting instructions). Increment cycle repeats 222 times per second (455kHz÷2÷16÷16), increment value should be 4.5010989ms. Attention should be payed to DPT digit, which is hexadecimal (no BCD correction). Also R5 register should be always set to correct value to display seconds and milliseconds correctly after ALU instruction.

```
;initialization
1800:   01D8     MOV      A.ALL,#0
        01DB     MOV      B.ALL,#0
;display mask to correctly display last digit
        07DB     MOV      B.EXP,#-1     ;#99
        01DE     MOV      D.ALL,#0
;4.50ms step value
        0A47     MOV      R5,#4
        02F6     MOV      D.DPT,R5
        0176     SHL      D.ALL,D
        0A57     MOV      R5,#5
        02F6     MOV      D.DPT,R5
        0176     SHL      D.ALL,D
;stopwatch is not running here
;wait for key press
;R5 contains DPT (decimal point) position
180A:   0A57     MOV      R5,#5
        0A30     WAIT     D3
;test CLR key
        087F     KEY      7F
        1804     BRA1     +2            ;180F
;clear counter
        01D8     MOV      A.ALL,#0
;test R/S key
180F:   0AA0     WAIT     D10
        08FD     KEY      FD
        180F     BRA1     -7            ;180A
;stopwatch is running here
;wait for R/S key released
1812:   01B0     ADD      A.ALL,A,D
        0A57     MOV      R5,#5
        1002     BRA0     +1            ;1815
1815:   0AA0     WAIT     D10
        08FD     KEY      FD
        100B     BRA0     -5            ;1812
;stopwatch is still running here
;wait for R/S key pressed
1818:   01B0     ADD      A.ALL,A,D
        0A57     MOV      R5,#5
        1002     BRA0     +1            ;181B
181B:   0AA0     WAIT     D10
```

```
        08FD     KEY      FD
        180B     BRA1     -5            ;1818
;stopwatch is not running here anymore
;wait for R/S key released
181E:   0AA0     WAIT     D10
        08FD     KEY      FD
        1005     BRA0     -2            ;181E
        182F     BRA1     -23           ;180A
```

DPT test

This example is little bit tricky. It changes R5 for every digit displayed so DPTs are displayed for every digit position.

```
1800:   0AF0     WAIT     D15
        0AD7     MOV      R5,#13
        0AC7     MOV      R5,#12
        0AB7     MOV      R5,#11
        0AA7     MOV      R5,#10
        0A97     MOV      R5,#9
        0A87     MOV      R5,#8
        0A77     MOV      R5,#7
        0A67     MOV      R5,#6
        0A57     MOV      R5,#5
        0A47     MOV      R5,#4
        0A37     MOV      R5,#3
        0A27     MOV      R5,#2
        181B     BRA1     -13           ;1800
```

7-segment decoder test

This test is also little bit tricky. It uses “bad” digit synchronization to display DPT on LED. As DPT is hexadecimal, this allows to display all combinations available in 7-segment decoder. Value displayed is BB.AAX, where BB is value in B.DPT, AA is value in A.DPT and X is resulting 7-segment digit.
Because of illegal synchronization, some keys behave strange! (RCL makes CPU reset, LRN row doesn't work, BST row doesn't detect key press but increment is done until keys are held – this can be useful to test higher values...)
Note: B.DPT can't be used to increment using mask value because B is on the same ALU input as mask constant, so B (or D) is or-ed with this constant instead of adding it.

```
1800:   0A01     CLR      IDL
        01D0     MOV      A.ALL,#0
;prepare digit mask to B register: 99990
        01D3     MOV      B.ALL,#0
        07DB     MOV      B.EXP,#-1
        0153     SHL      B.ALL,B
        0153     SHL      B.ALL,B
        07DB     MOV      B.EXP,#-1
        01D4     MOV      C.ALL,#0
;set IDLE but shifted so DPT is visible
        0AC0     WAIT     D12
        0A09     SET      IDL
;main loop
; wait for key press
180A:   0AE0     WAIT     D14
        0820     KEY      20
        180C     BRA1     +6            ;1812
        0A45     TST      KR[4]
        0A55     TST      KR[5]
        0A65     TST      KR[6]
        0A75     TST      KR[7]
        100F     BRA0     -7            ;180A
;check key press = debounce key
1812:   0AE0     WAIT     D14
        0820     KEY      20
        1805     BRA1     -2            ;1812
;increment A.DPT
        0300     ADD      A.DPT,A,#1
        1806     BRA1     +3            ;1819
;increment C.DPT if carry
        0324     ADD      C.DPT,C,#1
;and copy C.DPT to B.DPT
        0223     ADD      B.DPT,C,#0
;copy A.DPT to display the number
1819:   01D6     MOV      D.ALL,#0
        0206     ADD      D.DPT,A,#0
        0176     SHL      D.ALL,D
        0630     ADD      A.EXP,#0,D
;copy C.DPT to display the number
        01D6     MOV      D.ALL,#0
        0226     ADD      D.DPT,C,#0
        0176     SHL      D.ALL,D
        0176     SHL      D.ALL,D
        0176     SHL      D.ALL,D
        0930     ADD      A.MANT,#0,D
;clear COND
        1002     BRA0     +1            ;1824
;set DPT position
1824:   0A37     MOV      R5,#3
        1837     BRA1     -27           ;180A
```

Undiscovered secrets

List of unclear or still hidden things...

- WAIT BUSY opcode function is still unknown – never used in TI-58 ROM
- Strange that KR[7] is not used for constant addressing (SCOM constant ROM)
- RAM addressing produces some unclear digits when accessing address higher than 99. No idea if it is by-product of address calculation only.
- Unsure about COND bit for NO-OP and R5→adder ALU operation
- SAC DC-59: how it works?

Reference

U.S.pat 3900722

CPU a SCOM description; easy calculator implementation example including ROM dump (2x 1KB SCOM).

U.S.pat 4153937

Second ROM description – library storage and access description. ROM double SCOM and First ROM dump for TI-58 (unreadable!). Program codes. Printer codepage (Table VII).

U.S.pat 4006455

Magnetic card reader description.

U.S.pat 4020465
Printer description.
TI-5x service manual
Internal service information, TI-59 schematics, printer operation principles, RAM test program.
AR magazin, Construction appendix, 1985, pages 60-65
Schematics for TI-58, TI-58C, TI-59, PC-100A

Credits

Thanks to Hrast for first functional emulator for TI-58/59 and printer (www.hrastprogrammer.com/emulators.htm)