PROGRAMMING FOLDER S2000

This folder explain how to use programming software for S2000C, to have available calculus functions.
If you don't have a particular result, contact us and ask to have explains.

1. INTRODUCTION

S2000 has a microprocessor with a serial gate, some analog and digital inputs and outputs.(see later features).
It has a resident software (it means non modify , it is possible only if you substitute EPROM), it can do a certain number of operations stored in an EEPROM (Electrically Erasable Read Only Memory).
Executing operation type is fixed (chapter 3 describes them), while their number is function of EEPROM's loading. Normally one instruction loads one byte while one number loads 4 byte; maximum bytes is 240.
Operations to do at first are written by a PC in a file, by text editor "EDIT" standard in MS-DOS, where it is possible to modify, store and call-up as a normal text file.
See DOS problem folder to use editor.
You can use any other text editor if it doesn't insert any control characters type into the file.
It is so impossible to use editor like WORD or WORD STAR as documents.
Later we do compiler and instructions' transfer using S2000 program compiler ,optional with its own problem folder and cables.
Compiler controls that all instruction are correctly written, it codes them in a code that can be understand by S2000 and then it transfers them to S2000 by serial gate, verifying they are received without errors.
Compiler lists all compiled instructions; in case of keystroking error , compiler will write on video error type and errated instruction indication.
Compiler can control if instructions are written orthographically correct but it can't verify if their sequence is logically right.
During S2000's transfer can be some errors that are shown on video. In this case we have to repeat the all procedure.
After program's transfer, S2000 starts program execution and if there are executing problems it send them to PC to display them on video.
That is the reason why we test program inside S2000, testing all possible combinations for input values, it is necessary to have PC connected to S2000 and S2000C program working.
S2000 can't verify if instructions are logically corrected; it can only indicate if it finds a zero division or a similar error.

2. PROGRAMMING

Programmation (instructions' sequence) is similar to HP calculators, i.e. it works with stack pointers manipulated by given instructions.
Stack pointer is a filing cabinet where data are put or remove as the instruction we have to do; it is like a table where on it sheets are lean - in pile - and on them there are numbers.
Last sheet (or sheets) is the first to be used in operation to do (as LIFO queue).
When operation is done, on stack we have the result (or results) of this.
An example: how to sum two numbers.
Operations' sequence is:

1.2647	(line 1)
3.6744	(line 2)
+	(line 3)
STOP	(line 4)

N.B. Instructions can be written sequentially, putting a space between the two instructions.

In detail we see how data move on stack; on left are put instructions that are solved and on right the effect product on stack.
Those schemes don't appare on video, they are given only to explain.

Instruction	stack
no one	no one
1.2647	1.2647
3.6744	3.6744 1.2647
+	4.9391

Note:
- A number is an instruction too.
- Stack's base is underlined.
- At the beginning stack is empity so that every calculus operation is not executed and it will do executing error (S2000 detects it, not compiler).
- STOP instruction is executed by compiler;not by S2000. See list for more details.

It is obvious that stack's use is very convenient: it can substitute brackets used in common scientific notation.
An example: 1.25 * (100 - 20).
General rule, as we haven't brackets, we have to do firstly operations inside brackets.

Instruction	stack
100	100
20	20 100
-	80
1.25	1.25 80
*	100

Note:
- This way to do reduces memory used to store instructions.

Stacks can so contain partial calculus results, so that an operation like this is easy to do : 2 * 7 - 5 / 2 - 15.52

Instruction	stack
2	2
7	7 2
*	14
5	5 14
2	2 5 14
/	2.5 14
-	11.5
15.52	15.52 11.5
-	-4.02

Now we have to insert input and output variables too, not only constants otherwise equipment is no use.
Variables are only operations like +, - , the only difference is that they have other symbols or names and they modify or put values on stack.
For example we see the use for analog input and analog output.

Instruction	stack
AI1	0.953 (mesured)
2	2 0.953
/	0.4765
AO1	n.nnnn

As seen AI1 instruction leave on stack reading value on its analog input, this value is divided for 2 and then is used A01 instruction that send calculated value to analog output 1.
This instruction remove from stack using number, so the number at the top of the stack is the preexisting number at the beginning of calculation, so it is unknown.

Stack can have 10 numerical values max., if you insert more than 10 values, those on bottom are looses, while if an instruction doesn't have on stack all necessary numbers, it will take random numbers.

There are also instructions that uses or determines "digital" results.
Digital result is a number that is FALSE if 0, TRUE if different from 0.
It can be used even in instructions operating on normal numbers, and on contrary: operations waiting for digital values uses number as it is digital, that is to say equal or different from zero.
Operations that give a digital value, give 0 or 1.
For example COMP instruction, that given 2 inputs gives back 1 if the first input is higher than the second.

instruction	stack
AI1	0.725
5e-1	0.500 0.725
COMP	1.000
AI2	0.098
0.100	0.100 0.098
COMP	0.000

Note:
- Numbers can be introduct even in scientific notation in formats:
  +/-n.nnnnnne+ /- nn or +/-n.nnnnnn.
  Example: Valid numbers : 1.1278 , 0.00345, -4.28e-3, 10000, 1e4, 7848.33.

This example allows to reset a measure (AI1) if it is less than a determined value, now 0.200.

instruction	stack
AI1	0.123
DUP	0.123 0.123
0.200	0.200 0.123 0.123
COMP	0.000 0.123
*	0.000

In this eample we used a new instruction: DUP.
DUP duplicates value at the top of the stack.
The last operations reset AI1 values or leave it as it is.

By an opportune basic operation combinations , that are listed, it is possible to do lots of calculations and choose on analog or digital input or output.
Program is executed from the beginning to the end without interrupts except when verify errors on execution.
At the end of instructions' list it is necessary to put STOP INSTRUCTION; it is a control for compiler, that now compiles a list and transmits it to S2000 if you give transmission's option.
Total cicle's time depends from the number and type of executed instructions.

3. INSTRUCTIONS' LIST

This list describes what operations do; near the name are indicated values that the operation has to find on stack those that it leaves on stack: (n □ n1, n2, n3, ecc.); at arrow's left there are numbers that remain on stack at the end of operation, at right there are numbers necessary to work ; n1 is the first introducing number and so on , this is the reason why n1 is at the bottom of the stack.
Considered digital values are distincted by 'd'.
Are indicated also non admitted values (for any operations) : those values give an executing error that interrupts operations doing and declares error throught PC. S2000 restart from the beginning of the program.

+ (Sum) (n ⇐ n1 + n2)
it sums two numbers, substitute the two values with result.

- (Subtraction) (n ⇐ n1 - n2)
it subtracts the number at the top of the stack from the number that is down.

*** (Multiplication)** (n ⇐ n1 * n2)
it multiplicates two numbers.

/ (Division) (n < n1 / n2)
it divides two numbers. ERROR if n2 = 0.

NEG (n ⇐ -n)
it negates the number on the stack.

DUP (n, n ⇐ n)
it duplicates the number on the stack.

DEL (out ⇐ n)
it eliminates the number on the stack (it doesn't mean reset).

SWAP (n ⇐ n)
it exchanges the two numbers at the top of the stack, the top number becomes the second and the second the first.

RADQ (n ⇐ n)
it gives square. ERROR if n < 0.

LOG (n ⇐ n)
it gives Log . ERROR if n <= 0.

LN (n ⇐ n)
it does log 'e' base. ERROR if n <= 0.

EXP10 (n ⇐ n)
it does exponent n of 10.

EXPE (n ⇐ n)
it does exponent n of 'e'.

EXP (n ⇐ n1, n2)
it does exponent n2 of n1 . ERROR if n1 < 0.

COMP (d ⇐ n1, n2)
it compares two numbers on the stack, if n1 > n2: d = 1.

SWITCH (n ⇐ n1, n2, d)
it chooses two numbers as a digital number status: if d = 0 then n = n1; if d = 1 then n = n2.

NOT (d ⇐ d)
it negates d: if d = 0 then d = 1; if d = 1 then d = 0.

AI1, AI2, AI3, AI4 (n ⇐ analog input specificated)
it puts at the top of the stack reading value by its own analog input , range -1.000 and +1.000 corresponding an input signal -20....+20mA. Measure can be till +/-1.0240 ,2.5 resolution.ZS instruction must follow to read a signal 4..20mA.

AO1, AO2 (analog output specificated < n)
it copies the top value on analog output. Number is in 0-+1.000 range, it gives an output current 0-20mA range. Conversion is done till 1.0240 max. It must be preceded by ZA instrucion to have an output 4..20mA range.

ZA (n ⇐ n)
it moves zero in range 0 - 1 from 20%.
It is necessary to have 4..20 mA from analog outputs when n is in range 0.000 - 1.000.

ZS (n ⇐ n)
it subtracts zero from 20% to a numerical value in range 0 to 1.
It is necessary to read analog inputs in 4..20 mA.

DI1, DI2 (d ⇐ specificated digital input)

it puts on stack digital value read from specificated input : 0 if input is open, 1 if input is closed.

DO1, DO2 (specificated digital output ⇐ d)

it copys value at the top of stack on specificated digital output. 0 value opens circuit while 1 closes it.

STO0, STO1 etc up to STO4 (specificated register, n ⇐ n)

it stores in specificated register the value at the top of stack but it doesn't remove it from the stack. Registers from 0 to 4 are aviable to store variables for any use.

RCL0, RCL1 etc up to RCL4 (n ⇐ specificated register)

it recovers from specificated register value contained and puts it at the top of the stack.

It is complementar instruction to STO0 etc.

%...% it isn't an instruction but it delimitates comments. Between types % can be introduced a sentence variable length , compiler doesn't consider it.

4. SPECIAL INSTRUCTIONS

Those instructions have particular incomes we described later.

ANALOG INPUT INTEGRATOR

Following instructions reguards working operations of two analog input integrators that will give 100ms pulses, fixed duration, on digital outputs.
Max output frequence is 5Hz.
1 and 2 integrators make pulse to be on 1 or 2 output.
To activate integrators is enought to put into register pulse's value (PVAL) a number in the range 10000 to 4 billions (2e9).
It is necessary to give input's variable to integrator, it can be a calculus result and it must be in 40000 to 4 billions range(at full scale) to have precision not minor to the declared one.
If PVAL1 = 40000 and IVAL1 = 40000 output gives 1 pulse per second.
Example : we want to have an energy (one pulse = 1 Mcal) mesuring temperature difference between round trip and water pipe's capacity . AI1 is capacity in mc/h, AI2 is high temperature, AI3 is low temperature. Their own full scales are 2l/s, 0-50□C, 0-50□C.
Maximum capacity is in Kcal/s : 2 * 50 = 100. We multiplicate this capacity per 1000 to be in admitted range, we put this value in IVAL1.
Now we have to determine PVAL1value : we have capacity in Kcal , we want to have one pulse for 1000 Kcal. As we multlipicate per 1000 to be in range, we do the same thing with PVAL, we have 1000000.

instructions	comment
1000000	pulse's value
PVAL1	
AI2	
AI3	
-	inputs difference
50	full scale temperature
*	
AI1	
*	
2	full scale capacity
*	
1000	range setting
*	
IVAL1	
STOP	

PVAL1, PVAL2 (registrer pulse's value $\Leftarrow$ n)
it stores in register pulse's value. ERROR if n < 0.

IVAL1 , IVAL2 (input register integrator $\Leftarrow$ n)
it transfers to integrator variable to be integrated value. ERROR if n < 0.

ITYPE (integrators register type ⇐ d)

it transfers to register TIPO INTEGRATORI digital variable d.

When digital variable d is 0,integrators' type is the one see before,that is to say analog input ; when d is different from 0, integrators type is the following one , digital input.

It is impossible to have a digital integrator and an analogue one contemporary.

DIGITAL INPUT INTEGRATOR

This integrator counts input pulses (DI1 e DI2) and multiplicates them per an analog variable IVAL1, IVAL2.
A pulse in input DI1 will be multipied for IVAL1 and then sums to previous. When pulse's value will be reached there will be a pulse at output. Pulses at input can have a 20Hz frequency while those at output can have at maximal 5Hz frequency. Same indications given for analog type for PVAL and IVAL range.

IVAL1, IVAL2 (integrator input register ⇐ n)

it transfers to integrator variable's value to be multiplied to inputs pulses. ERROR if n < 0.

5. HOW TO USE ' S2000C.EXE' PROGRAM

S2000C compiler converts text file in intelligible form from S2000.
To launch the only program compilation without transfer it to S2000 keystroke : **S2000Conomefile.ext<invio>.**
The writing <invio> means push button INVIO on keyboard, this button is also named ENTER or RETURN, while o means you have to put a space.
If present, errors and their own codes will be printed.
To send a file to S2000, add control /t .
Example : **S2000Conomefile.exto/t<invio>.**
S2000C program remains on till **ESC** button is pushed, it allows to display error's messages received by S2000.
It is possible to launch program to have only error's messages from S2000, without compiling or trasfering any program.
To do this is enought to give **S2000C<invio>**, not labelling file.
In floppy disc you will find a file program demo **pro.txt.** Try to launch program as follow : **S2000Copro.txto/t<invio>** after PC is connected to the equipment (serial gate COM1) and to power supply mains.
After seconds on video there will be the writing : **trasmissione al S2000** follows : **trasmissione terminata** . This means operation is done correctly.
If not appares : **trasmissione terminata,** pushr **ESC** button and repeat operation till you have the described result.

6. ERROR CODES

When program is launching or executing, there can be some errors that are send to video as error codes ex. E05,#E02
Following list gives what each error code means.

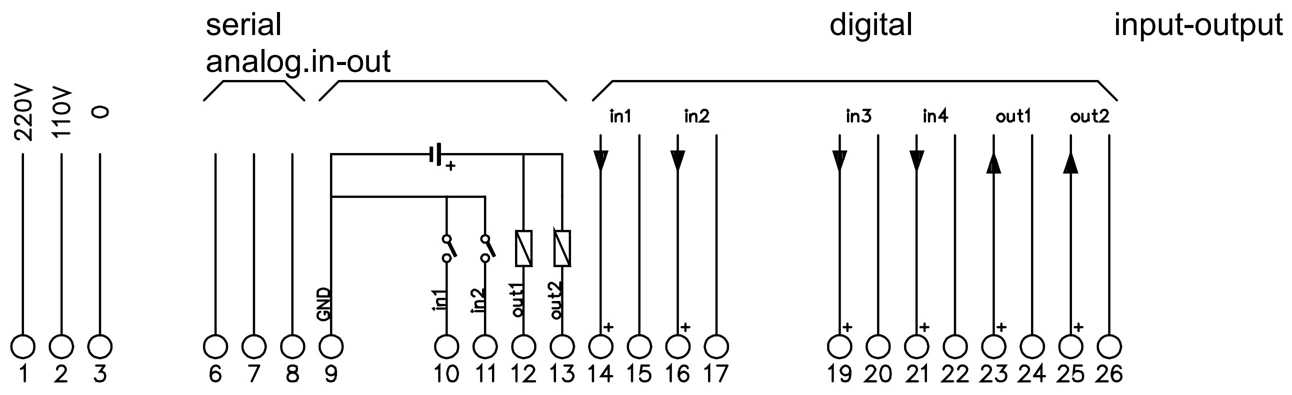
Error code	means	how to do
#E01	Transmission error	retry transmission
#E02	protocol error	as before
#E03	interrupted transmission	Control cable's integrity
E04	Reserved	
E05	Divide per zero	see again instructions'sequence
E06	negative radicand	as before
E07	Reserved	
E08	Negative number or zero logarithm	as before
E09	negative base exponential c.s.	
E10	eeprom's codes errated	Restart program; if error remains EEPROM should be damaged
E11	Reserved	
E12	negative pulse value	as before.

N.B. : The first three values has # , they can be only when program is transmitting to S2000.

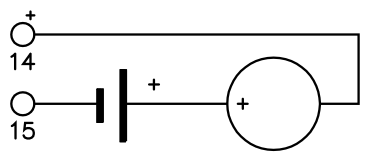
7. FEATURES

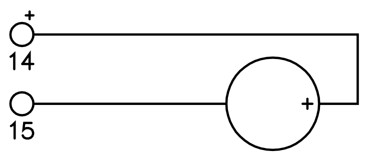
- Digital inputs : open contact voltage 5V, closed contact current 1mA.
- Digital outputs : max 300mA , 30Vdc.
- Analog inputs : 0..20mA or 4..20mA input resistance 100 ohm.
- Analog outputs : 0..20mA or 4..20mA active, maximum load resistance 300 ohm.
- Power supply : 110 o 220 Vac +- 10% 50 or 60 Hz, 3.5 VA.
- Temperature and humidiity : 0..+50°C, 90% at 40°C non condensing
- Box : to be coupled on 35mm bar (DIN46277) or fastening by screw, made by auto extinguish noryl.
- Size and weight : 157.5 x 95 x 69 mm, 500 g
- Analog convertion : at 12 bits input and output (4096 spot resolution)
- total precision better than 0,3 %.
- convertion's speed 150 mS each channel.
- analog and digital inputs not isolated (negative on common).
- green LED to signal if power.
- red LED to signal bed working (blinking).
- iRS232-C interface optocoupled to programming.

8. JUNCTION SCHEME

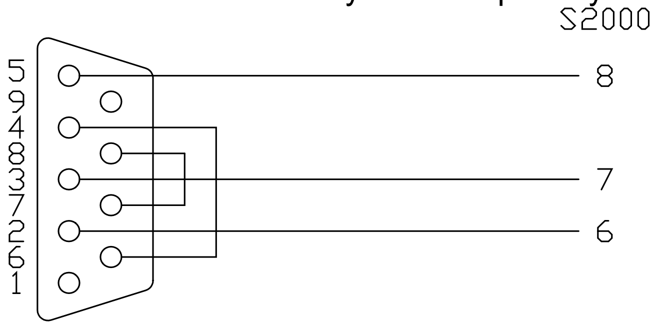


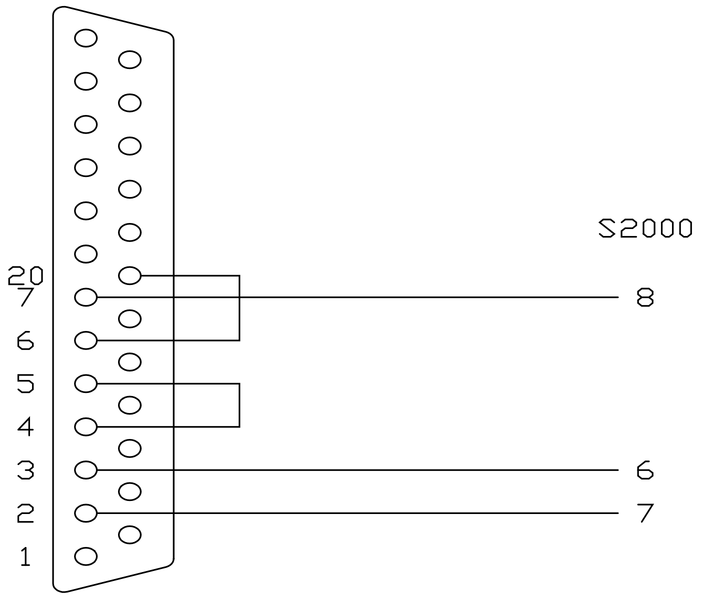
connection's detail for analg inputs :





Passive sensor connection. active sensor connection.
S2000 has RS232-C interface at 6,7,8 terminals.
This must be connected by a cable specially cabled as shown in pictures below.


S2000
5
9
4
8
3
7
2
6
1
8
7
6


S2000
20
7
6
5
4
3
2
1
8
6
7