SIGNAL PROCESSING LANGUAGE AND OPERATING SYSTEM

by

S. Weinstein
Naval Research Laboratory
Washington, D.C., U.S.A.

ABSTRACT Signal Processing Language/One (SPL/I) was developed at the Naval Research Laboratory (NRL) as a high-level computer programming language for signal processing. SPL/I is the first high-level language to be designed for real-time signal processing applications and selected for production use in the U.S. military.

Signal Processing Language/One (SPL/I) is a high level computer language developed primarily for use in signal processing. The language was developed at the Naval Research Laboratory (NRL) and is the first high-level language to be selected for production use in a real-time signal processing application by the United States military. The language is a very flexible block structured language. It contains a wide variety of flow-of-control statements, a rich set of data declarations, and a thorough set of multiprocessing creation and synchronization primitives.

The multiprocessing, or more properly, the multiprogramming features, include process syncrhonization in the runtime environment through the use of counting semaphores and binary semaphores (SPL/I resource variables). SPL/I also provides the user with the ability to terminate and activate processes.

The flow-of-control statements in SPL/I include the definition and invocation of valued and non-valued procedures, the WHILE, UNTIL, and FOR looping statements, the conditional IF-ELSE statement, and the alternative selection CASE statement. The remaining executable statements include an assignment statement, limited GOTO, structured loop termination statements, and an empty statement

The language enables the user to manipulate standard primitive data types (e.g., integers and bit strings). The user can define and use extended data types consisting of groups of like data (ARRAYs) and groups of unlike data (STRUCTUREs) in a simple manner. These arrays and structures can be referenced by accessing single elements, or the entire structure or array. Subsections of arrays can be referenced by a method called array "slicing". The user can declare variables both of dynamic (AUTOMATIC) and static storage classes, and, under complete programmer control, can perform runtime allocation and deallocation of storage for variables. This flexibility allows the user to control and perhaps reduce storage requirements for many problems.

The commenting construct within SPL/I is one of nested square bracket pairs. This, along with the free format structure of the language, makes the flow-of control in SPL/I software easy to understand and encourages self documentation.

To enhance reliability in detecting program errors and again encourage self-documenting code, SPL/I requires explicit definitions of data types and variables, the compatibility of data types in operations, and explicit conversions between data types. These are all checked at compile time, thus eliminating a large class of possible error conditions.

SPL/I requires runtime support for the multiprocessing and dynamic storage features of the language. To prevent the proliferation of different executives for different SPL/I target computers, the Common Real-Time Operating System (CROS) was developed by NRL as the common SPL/I executive. In order to aid in the transportability, CROS is itself written 93% in SPL/I.

CROS offers a variety of process scheduling and dispatching facilities. The scheduling options include process creation on a demand basis, on a periodic basis, and in response to asynchronous events.

CROS is structured such that it is possible to select a subset of its many features that is right for a particular system; thereby eliminating storage and time penalties that would otherwise be incurred by unused language support.

Other features of CROS include high-level I/O and error handling capabilities. For system design evaluation, CROS offers historical data gather on event occurrences and timing information on sections of the application system.

After the user has coded part or all of a system the code is then run through the SPL/I Compiler where semantic and syntax checks are made and object code generated. This object code is then stored in a library by the SPL/I librarian. The user then runs the SPL/I Linkage Editor where the application system is built and a load image built. The user now has the opportunity to specify various attributes of the system and libraries to search to resolve external references. In addition, the Linkage Editor will choose the smallest version of CROS that will handle all the features the user has specified for the system. The user now can either run the system on a simulator or on the machine itself.

Production use of SPL/I with CROS began in 1977 involving two platforms running with the AN/UYS-1 Advanced Signal Processor; additional platforms will be added later. Effort is now complete for retargeting SPL/I and CROS for the Navy standard minicomputers AN/UYK-20 and AN/AYK-14.

The following pages of visual aid texts summarize the language and its application (Ed.)

DISCUSSION

J.E. Vernaglia Does SPL/I or CROS give the programmer any assistance in setting up control blocks for the storage transfer controller and arithmetic processor?

S. Weinstein Yes, and with the Modular Signal Processing Software Development System currently under development, the capabilities will be even further enhanced.

J.E. Vernaglia Did the conversion from assembly language to SPL/I include all P-3 processing modes, or just LOFAR?

S. Weinstein No, but more than just LOFAR.

Y. Lundh You said SPL/I and CROS lend themselves to multi-processing. How does SPL/I and/or CROS know what the multi-processor looks like?

S. Weinstein Currently, CROS does not address the multi-processor configuration.

SPL/I

A HIGH LEVEL LANGUAGE FOR DIGITAL SIGNAL PROCESSING

SPL/I OVERVIEW

BLOCK STRUCTURED

STRONGLY TYPED

SCALAR AND ARRAY OPERATIONS

FACILITIES FOR INDEPENDENT PROCESS EXECUTION AND CONTROL

FEATURES NEEDED FOR DIGITAL SIGNAL PROCESSING, GRAPHICS, AND SYSTEM PROGRAMMING

TARGET COMPUTER INDEPENDENT

DECLARATIONS

VARIABLE	Named memory location of a particular data type
MODE	Programmer-declared data type
PROCEDURE	Block of code invoked as a valued function or non-valued subroutine
PROGRAM	Block of code activated as independent process

DECLARATIONS

EXAMPLES

```
VAR A, B, C: INT;

MODE INT_ARRAY_3:  INT ARRAY(*,*,*);
VAR M:  INT_ARRAY_3 OF SIZE(3,4,5);

PROCEDURE SIN(X:  FLOAT INPUT):  FLOAT;
    [sine computation]
    •••
ENDPROCEDURE SIN;
```

VARIABLE DECLARATION

NAME	Name by which programmer refers to variable
STORAGE CLASS	Time of allocation of variable
MODE	Data type of variable
INITIALIZATION PHRASE	Optional initialization value upon allocation of variable

STORAGE CLASSES

AUTOMATIC	Storage allocated at block entry and deallocated at block exit
STATIC	Storage allocated before program execution and deallocated at program termination
GLOBAL/EXTERNAL	Static Allocation but accessible to multiple modules
ALLOCATED VARIABLES	Storage allocated dynamically under programmer control

PRIMITIVE MODES

ARITHMETIC

INT	Integer
DINT	Double precision integer
FLOAT	Real
FRAC	Fraction
CINT	Complex integer
CFLOAT	Complex real
CFRAC	Complex fraction

PRIMITIVE MODES

LOGICAL

BOOL	Logical (TRUE, FALSE)

STRING

BIT	Bit string
STRING	Character string

MULTIPROCESSING

PROCESS	Process identification
RESOURCE	Logical or physical resource

MODE DECLARATION

ARRAY	Homogenous data aggregate
STRUCTURE	Non-homogeneous data aggregate
POINTER	Pointers to data elements allocated under programmer control

PROCEDURE AND PROGRAM DECLARATIONS

NAME	Name by which procedure is invoked or program is activated as a process
FORMAL PARAMETER LIST	List of names and modes of formal parameters used by procedure or program
RETURN MODE	Mode of return value on valued procedures only
BODY	Declarations and statements of procedure or program

ASSOCIATIONS OF ACTUAL AND FORMAL PARAMETERS

VALUE	Formal parameter is copy of actual parameter value at time of invocation.
INOUT	Formal parameter refers to storage location of actual parameter.
INPUT	Formal parameter refers to storage location of actual parameter. Cannot be modified.

OPERATORS

ARITHMETIC

**	EXPONENTIATION
+,-	UNARY PLUS, MINUS
*,/	MULTIPLICATION, DIVISION
+,-	ADDITION, SUBTRACTION

STRING

#	CONCATENATION

OPERATORS

RELATIONAL

=,/=,<,>,<=,>=

LOGICAL

NOT	LOGICAL NOT
AND	LOGICAL AND
OR, XOR	LOGICAL OR, EXCLUSIVE OR

VARIABLE REFERENCE

@	ALLOCATED VARIABLE REFERENCE
.	STRUCTURE COMPONENT REFERENCE
( )	ARRAY ELEMENT REFERENCE

EXPLICIT MODE CONVERSIONS

```
VAR A, B :  STATIC INT:
VAR C :  AUTOMATIC FLOAT:
   •
   •
   •
A : = B + INT(C);
```

ARRAY AND SCALAR ARITHMETIC

[SCALAR:]

```
VAR I, J, K: AUTOMATIC INT;
  •
  •
  •
I := J + K;
```

[ARRAY:]

```
MODE INT_ARRAY_2: INT ARRAY(*,*);
VAR A, B, C:
    INT_ARRAY_2 OF SIZE(6,8);
  •
  •
  •
A := B + C;
```

ARRAY SLICING

```
MODE INT_ARRAY_3:  INT ARRAY(*,*,*);
MODE INT_ARRAY_2:  INT ARRAY(*,*);

VAR AR1:  INT_ARRAY_3 OF SIZE(10, 10, 10);
VAR AR2:  INT_ARRAY_3 OF SIZE(10, 10, 5);
VAR AR3:  INT_ARRAY_2 OF SIZE(10, 5);
    •
    •
    •
AR2 := AR1 (*,*,3 THRU 7);
AR1(*,1, 5 THRU 8) := AR2(*,3, 2 THRU 5);
AR2(9,*,*) := AR3;
    •
    •
    •
```

EXECUTABLE STATEMENTS

DATA MOVEMENT AND COMPUTATIONAL
- *ASSIGNMENT*

PROCEDURE/CONTROL
- *PROCEDURE INVOCATION*
- *RETURN*

BLOCK/CONTROL
- *BEGIN*
- *EXIT*

CONDITIONAL
- *IF-THEN-ELSE*
- *CASE*

EXECUTABLE STATEMENTS

LOOP/CONTROL
- *WHILE*
- *UNTIL*
- *FOR*
- *EXITITERATION*
- *EXITLOOP*

OTHER
- *EMPTY*
- *GO TO*

EXECUTABLE STATEMENTS

PROCESS MANIPULATION

ACTIVATE
TERMINATE

PROCESS SYNCHRONIZATION

REQUEST
RELEASE

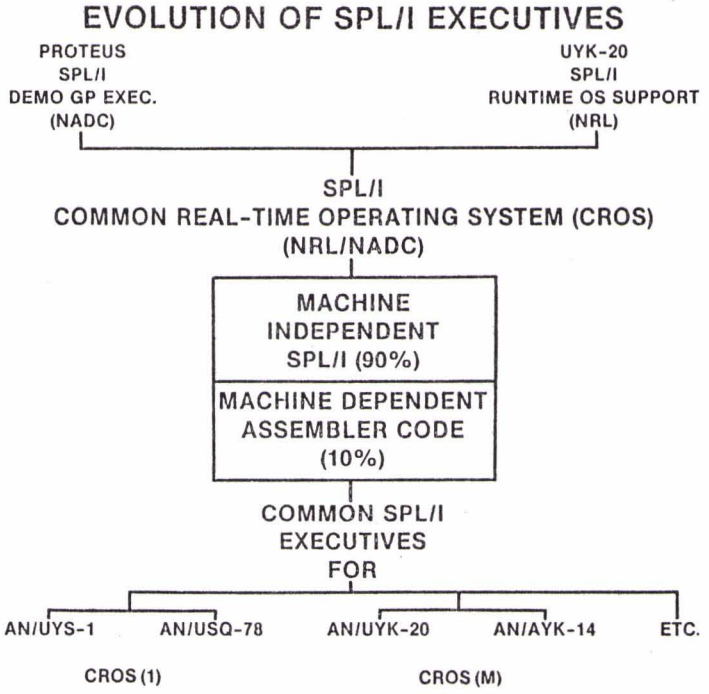
CROS

COMMON REAL-TIME OPERATING SYSTEM
FOR
SPL/I

CROS MOTIVATION

- NEED FOR EFFICIENT SUPPORT OF SPL/I "MULTIPROCESSING" STATEMENTS AND FACILITIES

- POTENTIAL FOR PROLIFERATION OF SPL/I EXECUTIVE ENVIRONMENTS FOR DIFFERENT TARGET COMPUTERS AND TARGET COMPUTER CONFIGURATIONS



CROS FEATURES

- VARIETY OF SCHEDULING AND DISPATCHING FACILITIES
- ACCESSIBILITY OF FACILITIES FROM BOTH SPL/I AND ASSEMBLY LANGUAGE APPLICATIONS PROGRAMS
- EXTENSIVE SET OF OPTIONS FOR SUPPORT TRADEOFFS AND REQUIREMENTS
- PRE-RUNTIME SPECIFICATION AND GENERATION OF PROCESS ATTRIBUTES FOR RUNTIME EXECUTION SAVINGS
- AUTOMATIC SELECTION OF SMALLEST VERSION WITH REQUIRED FACILITIES

CROS DESIGN GOALS

- OPERATING SYSTEM ENVIRONMENT FOR ALL SPL/I TARGET COMPUTERS—INITIALLY AN/UYS-1, AN/USQ-78, AN/UYK-20, AN/AYK-14
- SUPPORT FOR A WIDE RANGE OF OPERATING SYSTEM FACILITIES FOR TACTICAL SIGNAL PROCESSING AND "REAL-TIME" APPLICATIONS
- TARGET MACHINE-INDEPENDENT USER INTERFACE
- TARGET MACHINE-INDEPENDENT IMPLEMENTATION
- EXTENSIBILITY OF I/O FACILITIES

CROS PROCESSES

- SCHEDULING:
  - ON DEMAND BY ANOTHER PROCESS
  - PERIODICALLY
  - ON EVENT OCCURRENCE
- DISPATCHING:
  - PRIORITY-DRIVEN, 255 LEVELS
  - USER-SELECTED POLICY:
    "NORMAL" - RUN TO COMPLETION OR BLOCKAGE
    "ROUND-ROBIN" - PROCESSOR ALLOCATED EQUALLY
- COORDINATION:
  - VIA SPL/I LANGUAGE FACILITIES

CROS SERVICE ROUTINES

- PROCESSES:
  ACTIVATE, TERMINATE, ENABLE, DISABLE, SUSPEND, UNSUSPEND
- RESOURCES:
  REQUEST, RELEASE
- COUNTING SEMAPHORES:
  INITIALIZE, P, V
- FREE STORAGE:
  ALLOCATE, DEALLOCATE

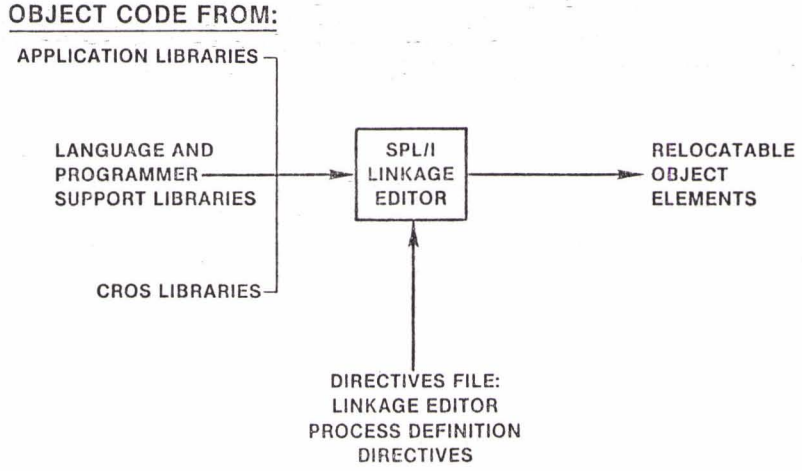
CROS EXECUTIVE SERVICE ROUTINES
(CONTINUED)

- HIGH LEVEL INPUT/OUTPUT
- ERROR HANDLING
- PERFORMANCE MONITORING
- EVENT RECORDING/TIMING

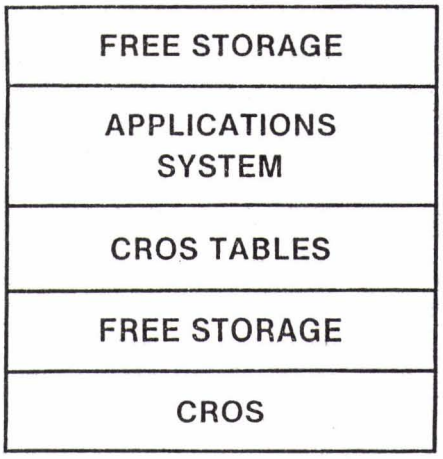
CROS I/O

- FIRM INTERFACE SPECIFICATION FOR I/O DEVICE DRIVERS:
  - REQUEST I/O ROUTINE
  - WAIT ON I/O ROUTINE
  - HALT I/O ROUTINE
  - I/O INTERRUPT ROUTINE
- HIGH LEVEL (SPL/I) USER I/O PRIMITIVES
  - OPEN/CLOSE
  - READ/WRITE
  - CONTROL

CROS LINK EDIT TIME



CROS RUN TIME ORGANIZATION



LINKAGE EDITOR PROCESS DEFINITION DIRECTIVES

- DEFINE CROS FACILITIES REQUIRED BY SYSTEM
- DEFINE NAMES AND CHARACTERISTICS OF SYSTEM
- DEFINE SPECIAL PROCESSING REQUIREMENTS OF SYSTEM

CROS CONFIGURATION OPTIONS

- MULTIPROCESSING/NONE
- PERIODIC PROGRAMS/NONE
- SELECTED EVENT PROGRAMS/NONE
- TERMINATIONS/NONE
- RESOURCE VARIABLES/NONE
- SEMAPHORES/NONE
- SYNCHRONIZATION VARIABLE IMPLEMENTATION OPTIONS
- HISTORY OPTIONS
- ERROR HANDLING OPTIONS