TO: MSPM Distribution
FROM: J. H. Saltzer
SUBJ: BK.4.01 reissue
DATE: 10/02/67

The old processor communication table has been expanded and renamed the system communication segment. It now contains all configuration dependent constants used in intercommunication among major system hardware modules. This version of BK.4.01 replaces the section entitled "the processor communication table" and originally numbered BK.1.04.

Published: 10/02/67
(Supersedes: BK.4.01, 06/02/66)

Identification

System Communication Segment
Chester Jones, L. J. Lambert, Harlow Frick

Purpose

The System Communication Segment (SCS) is a per system data base shared by all processes running under the same version of Multics. SCS contains most variables which are hardware configuration dependent and is built during system initialization. SCS is a wired-down, read-only data segment except during initialization or system reconfiguration. It is accessible only in the hardcore ring.

Contents of SCS

SCS contains the following information:

1.) Interrupt handler pointers, argument space and mask patterns.
2.) Processor SMIC patterns.
3.) Processor system controller pointers.
4.) Processor signalling connect flags.
5.) Number of processors.
6.) Processor port connection table.
7.) GIOC port connection table.
8.) Drum port connection table.
9.) Interrupt level masks.
10.) Drum device indexes.
11.) System clock device indexes.

Information in SCS will next be described in more detail. PL/1 declarations are included with each description.

(1) Interrupt pointers, argument space and mask patterns

This section describes information required for each of the 32 interrupt cells in a system controller. Each of the following PL/1 declarations defines 32 items. The first is for interrupt cell 0, the second for interrupt cell 1, etc.

```
dcl   scs$int_hlr (0:31) ptr ext;
```

These ITS pairs point to the interrupt handler to be called when this interrupt occurs.

```
dcl   scs$argument (0:31) bit(36) ext;
```

These items contain data used to construct argument words passed to the interrupt handler when it is called. The argument is broken into two 18 bit fields. The most significant 18 bits contains the active device index and the least significant 18 bits contains the interrupt number. For example, for status channel 3 of the GIOC assigned index 1, the active device index and interrupt numbers would be 1 and 3, respectively.

```
dcl   scs$mask_pattern (0:31) bit(72) ext;
```

These interrupt masks are the value loaded to the AQ register prior to execution to the initial SMCM instruction when the respective interrupt occurs.

(2) Processor SMIC patterns

SCS contains four 36-bit patterns needed in order to produce interrupts. One of these 4 patterns is placed in the A register and then an SMIC instruction is executed in order to produce one of the three process interrupts.

PL/1 declarations for these patterns are the following:

```
dcl   scs$pre_empt_pattern    bit(36) ext;
dcl   scs$time_out_pattern    bit(36) ext;
dcl   scs$quit_pattern        bit(36) ext;
dcl   scs$sys_trouble_pattern bit(36) ext;
```

(3) Processor system controller pointers

SCS contains an array of seven ITS pointers which are used to address the system controller from which the processor expects interrupts. The first entry in this array is an ITS pointer which points to the system controller, via SCAS (BK.4.02), which transmits interrupts to the processor assigned index 1, the second entry to the system controller for the processor assigned index 2, etc. This array is used, for example, when masking interrupts in the interrupt interceptor, or when setting an interrupt cell to trigger a process interrupt. The PL/1 declaration for this array is as follows:

```
dcl   scs$proc_contr_ptr  (7) ptr ext;
```

(4) Processor signalling connect flags

SCS contains an array of seven entries called the connect flag array. The first entry is for the processor assigned index 1, the second for the processor assigned index 2, etc. Each entry serves as a signal between one of the processors in the system and another processor. (See BK.3.08 for a discussion of how the connect flag array is used.) Note that these flags are set and reset by master mode routines, thereby overriding the read-only attribute.

```
dcl   scs$connect_flag (7) fixed;
```

(5) Number of processors

The following PL/1 declaration is for an entry in SCS containing the number of processors. It is used, for example, by the procedure which issues connect instructions to processors (described in BK.5.05).

```
dcl   scs$nprocessors fixed;
```

(6) Processor port connection table

The following PL/1 declaration is for a table containing physical system controller ports which each processor is connected to. The first entry in the table is for the processor assigned index 1, the second for the processor assigned index 2, etc.

```
dcl   scs$processor_port (7)  bit(3);
```

The above entry is used by the procedure which issues connect instructions to processors (described in BK.5.05).

(7) GIOC port connection table

The following PL/1 declaration is for a table containing physical system controller ports which each GIOC is connected to. The first entry in the table is for the GIOC assigned index 1, the second for the GIOC assigned index 2, etc.

```
dcl   scs$gioc_port (4) bit(3);
```

The above entry is used by the procedure which fabricates GIOC connect operand words. The following call illustrates how this table can be used.

```
dcl  cow bit(36);

cow = left33bits||scs$gioc_port(index);

call master_mode_ut$cioc(cow);
```

(8) Drum port connection table

The following PL/1 declaration is for a table containing physical system controller ports which each drum is connected to. The first entry in the table is for the drum assigned index 1, the second for the drum assigned index 2, ect.

```
dcl  scs$drum_port (4) bit (3);
```

The above entry is used by the procedure which fabricates drum connect operand words. (Also called PCWs or Peripheral Control Words.) Note that scs$drum_port must be used with regard to one restriction: The procedure which fabricates the drum pcw must store the pcw (which is fabricated using the appropriate entry in scs$drum_port) into the same system controller as the base address of the drum, and use this prefabricated pcw as the argument when calling the cioc procedure. This is because of the drum hardware restriction that the pcw must be in the same system controller as the base address of the drum.

(9) Interrupt level masks

The following PL/1 declaration defines address space for masks which are used by the interrupt mask procedure. A more detailed definition may be found in BK.5.01.

```
dcl  (

     scs$sys_level,              /* all except emergency
                                    intervention */
     scs$gioc0_level,            /* GIOC status channel 0 and
                                    lower */
     scs$clock_trouble_level,    /* clock trouble and lower */
     scs$drum_level,             /* drum and lower */
     scs$drum_ctl_level,         /* drum control and lower */
     scs$drum_data_level,        /* drum data and lower */
     scs$drum_pgm_level,         /* drum program and lower */
     scs$gioc_level,             /* GIOC and lower */
     scs$gioc1_level,            /* GIOC status channel 1 and
                                    lower */
     scs$gioc2_level,            /* GIOC status channel 2 and
                                    lower */
     scs$gioc3_level,            /* GIOC status channel 3 and
                                    lower */
     scs$clock_pgm_level,        /* clock programmed interrupts
                                    and lower */
     scs$swap_level,             /* process interrupts only */
     scs$pre_empt_level,         /* pre-emption and lower */
     scs$time_out_level,         /* time-out interrupts and lower */
     scs$quit_level,             /* quit interrupts and lower */
     scs$open_level,             /* all interrupts allowed */
     scs$drain_pre-empt,         /* special mask to allow pre-emption
                                    interrupt only */
     scs$drain_time-out,         /* same for timer runout */
     scs$drain_quit              /* same for quit */

               ) bit(72) ext;
```

(10) Drum device indexes

Each drum has a device index assigned to each of its 3 interrupts. (See BQ.6.07.) The Drum Interrupt Handler translates drum interrupts into device indexes by referencing the following table:

```
dcl   1   scs$drum (4) ext,

        2 program_device_index fixed,
        2 control_device_index fixed,
        2 data_device_index    fixed;
```

(11) System clock device indexes

Each system clock has a device index assigned to each of its 2 interrupts. (See BQ.6.07.)

The Calendar Clock Interrupt Handler translates system clock interrupts into device indexes by referencing the following table:

```
dcl    1    scs$clock (3) ext,

            2 alarm_device_index   fixed,
            2 trouble_device_index fixed;
```