G0035

May 20, 1966

To: MULTICS Distribution

From: R. C. McGee

The attached paper by Nguyen, Slosberg, and Joel describes a method of using the 645 running under K2 GECOS for debugging MULTICS modules. The programming necessary for accomplishing this mode of operation is being done jointly by Dave Joel's Development Tool Unit in Cambridge and my Programming Integration Unit in Phoenix.

The method dedicates one GIOC for use by GECOS and leaves the second GIOC available for use by MULTICS modules. In addition, it provides a method for a module under test to gain complete control of all 645 hardware features while maintaining access to the development tools provided under GECOS.

/ch

May 13, 1966

Initial Utilization of the GE-645 at Project MAC

V. B. Nguyen, D. H. Slosberg, D. E. Joel

Introduction

It is planned to utilize the GE-645 in two distinct phases:

1. Phase 1 uses the hardware in exactly the same way that the GE-635 is currently being used, i.e., using the present 6.36 and 64.5 systems and their logical extensions.

2. Phase 2 brings the user closer to the GE-645 by removing the simulator from the 6.36 and 64.5 environments, allowing a process to execute instead of being simulated, permitting the process to perform input/output using the entire facility of one GIOC, and permitting the process to handle its own faults.

Phase 1 provides a good environment for "bringing up" the GE-645 using K1 GECOS (or later K2 GECOS), and at the same time giving complete backup in the shape of the GE-635 still on site. Note that K1 or K2 GECOS consists of GE-635 GECOS plus a supplement (herein named the GECOS Supplement).

Phase 2 provides the software developer with a broader facility, while still retaining basic control of the GE-645 by K1 or K2 GECOS.

A main design objective is that users of the 6.36 and 64.5 systems will be able to run jobs in Phase 2 without realizing the disappearance of Phase 1.

Phase 1

The only requirement to implement this phase is that K1 GECOS (using an IOC) be operational, i.e., that the GE-645 can be run in exactly the same way as a GE-635 (as viewed from the users end).

This environment is to be checked out in Phoenix prior to delivery of the GE-645. The checkout procedure to be used is complete running of typical 6.36 and 64.5 jobs.

Phase 2

The method of operation planned here is exactly the same as in Phase 1 except during the one activity in a 6.36 or 64.5 job which is devoted to loading and executing a 645 process.

The main characteristics of the operation at load/execute time are as follows:

1. The GECOS Supplement, on request, yields information about the '635 Slave' activity currently in execution. This allows the 645 loader to determine its own environment.

2. The GECOS Supplement, on request, sets a '645 Process' switch which causes interrogation of a users pseudo fault and interrupt vectors when a fault or appropriate interrupt occurs, and reflects the condition to the user as indicated by the contents of the appropriate vector.

3. When a fault condition occurs with the '645 Process' switch on, and the appropriate entry in the users pseudo fault vector indicates that the user is not handling this condition, the GECOS Supplement passes control to a termination routine in the 645 loader through a fixed communication zone. This permits the normal form of dump given by 6.36 or 64.5 to be obtained in preference to the GECOS abort dump.

4. It is normal practice to use two libraries to set up execution. One contains 635 subprograms (e.g., the 645 loader), and the other contains 645 assembled segments (especially the special inclusion segments). Any changes required converting from Phase 1 to Phase 2 operation are handled entirely by manipulation of these two libraries.

5. The 645 Process placed into execution is in a position to do anything it wants to, and thus can cause a catastrophic software failure. It is a design objective that the user have this freedom while still trying to catch error situations if possible (see 3 above).

The specific requirements of the system are described below in some detail, with particular emphasis being given to interface details:

1. The GECOS Supplement

When a 635 slave activity is set up, the basic mechanism is that of building a descriptor segment. The entries which are required in the descriptor segment are described in Figure 1.

Segment Number	Description	Descriptor Attributes
5	Describes the assigned 635 slave memory, unpaged, 1024 word blocks.	Slave procedure, slave access, write permit.
6	Describes the mailbox area for the GIOC assigned to 645 process work only, unpaged.	Data

Figure 1. Descriptor Segment for 635 Slave Activity.

It is necessary that the 645 loader make requests of the Supplement. The vehicle used is the Master Mode Entry 1 command, which is restricted to use in GE-635 software only. The specific requests which the 645 loader can make are as follows:

a) Return a descriptor word.

```
LDA     1,DL    Type 1
LDQ     N,DL
MME1    64
```

The N'th word in the descriptor segment (N starts at zero) is passed back by the Supplement in register A.

b) Change Mode, set '645 Process' switch.

```
LDA     2,DL
LDQ     = MVFDB18/FVECTR,18/IVECTR
MME1    64
```

FVECTR and IVECTR are locations in the '635 slave memory' at which the pseudo fault and interrupt vectors are located.

The Supplement changes the descriptor for segment 5 from slave procedure, slave access, write permit to master procedure.

The Supplement sets the '645 Process' switch so that if faults or interrupts (from the dedicated 645 process GIOC) occur, the pseudo vectors provided by the user are interrogated (see later discussion on fault and interrupt handling).

c) Discontinue interrogation of pseudo fault vector.

```
LDA   3,DL
MME1  64
```

This request is honored only if the '645 Process' switch is on. The function provided is the ability to have faults interpreted

by GECOS while executing 'escape' coding (which really is 635 code).

d) Resume interrogation of pseudo fault vector.

```
LDA    4,DL
MME1   64
```

This request is honored only if the '645 Process' switch is on.

When a fault occurs the Supplement checks a set of conditions and reacts accordingly. The specific checking is shown in the decision table diagrammed in Figure 2.

An example of the use of this diagram is: fault occurs, and execution mode at time of fault occurrence is relative, and '645 Process' switch is off, then follow ACTION #3.

When an interrupt occurs on a device dedicated to GECOS (on an IOC with K1, or on a GIOC with K2), the Supplement reflects the interrupt to GECOS. When an interrupt occurs on the GIOC dedicated to 645 process work, the Supplement checks a set of conditions and reacts accordingly. The specific checking is shown in the decision table diagrammed in Figure 3.

Figure 2. Fault handling in the GECOS Supplement - (Read from left to right)

Occurrence of a fault	Execution Mode Absolute at time of fault occurrence	Fault Control-unit move switch is on. (Fault occurred while a prior fault was being simulated)			ACTION #1: Communicate trouble (code 66) to slave program, and go to slave terminate routine (see definition of communication Region)
		Interrupt Control-unit move switch is on. (Fault occurred while an interrupt was being simulated)			ACTION #2: Communicate trouble (Code 65) to slave program, and go to slave terminate routine.
		Control-unit move switches are both off.			ACTION #3: Reflect the fault to GECOS
	Execution Mode relative at time of fault occurrence	'645 Process' switch is off.			
		'645 Process' Switch is on	Users pseudo fault vector is inactive (MME1 64 type 3 in effect)		
			User pseudo fault vector active	Instructions in users pseudo fault vector are SCU and TRA	ACTION #4: Turn on the fault control-unit move switch. Move the control-unit; this simulates the users SCU (a fault can occur here resulting in ACTION #1). Turn off the fault control-unit move switch. Execute the users TRA.
				Instruction in users pseudo vector are not SCU and TRA	ACTION #5: Communicate fault information to slave program and go to slave terminate routine.

crk 2.

Figure 3. Interrupt handling in the GECOS Supplement

(Read from left to right)

Occurrence of an Interrupt	Device on which interrupt occurred is dedicated to GECOS			ACTION #1: Reflect the interrupt to GECOS
	Device on which interrupt occurred is dedicated to 645 processor (GIOC)	'645 Process' switch is off	Instructions in users pseudo interrupt vector are SCUand RCU	ACTION #2: Ignore the interrupt
		'645 Process' switch is on.	Instructions in users pseudo interrupt vector are SCU and TRA	ACTION #3: Turn on the interrupt control-unit move switch. Move the control-unit; this simulates the users SCU (a fault can occur here resulting in fault ACTION #1). Turn off the interrupt control-unit move switch. Execute the users TRA.
			Instructions in user pseudo interrupt vector are not SCU and TRA or SCU and RCU	ACTION #4: Communicate interrupt information to slave program and go to slave terminate routine.

2. The 645 Loader and Associated Routines

When the 645 Loader gets control, it has the responsibility of setting up a descriptor segment for the 645 Process, loading several fixed segments (described in Figure 4) and all other requested segments (details may be found in writeups on 6.36 and 64.5), setting up a communication region for the purpose of communicating with the Supplement and the 645 Process segment 'escape', and finally transferring control to the 645 Process in segment 'init'.

The fixed segments which are loaded are as follows:

Segment Number	Description	Descriptor Attributes
4	Pseudo fault vector at 400_8 in '635 slave memory'. Segment name is 'fvectr'. 192 words long, unpaged. First 64 words are the fault vector, remainder reserved for ITS pointers for SCU-TRA instruction pairs. Loader initializes to zero.	Data
5	Describes the assigned 635 slave memory, unpaged, 1024 word blocks. Information obtained from Supplement (MME1 64). Segment name is 'memory'.	Slave procedure Slave access, write permit.
6	Describes the channel mailbox for the GIOC assigned to 645 Process work. Unpaged. Segment name 'giocl'.	Data
7	Pseudo interrupt vector at 700_8 in '635 slave memory'. Segment name is 'ivectr'. 192 words long, unpaged. First 64 words are the interrupt vector(s), remainder reserved for ITS pointers for SCU-TRA instruction pairs. Loader initializes to SCU, RCU instruction pairs.	Data

Figure 4. Basic entries in the descriptor segment for a '645 Process'

The communication region which the 645 Loader sets up has the following format:

Address (decimal) within '635 slave memory'.	Length (words)	Description	
240	2	Area which segment 'escape' uses to pass a pointer (ITS pair) to the argument list.	
242	2	Area which segment 'escape' uses to save control information. 635 escape coding returns to '645 Process' by executing an RTD 242 instruction.	
244	6	Area used by the Supplement to hold the control unit when the Supplement has determined that the '645 Process' should terminate itself.	
250	1	Area used by the Supplement to indicate the fault number or interrupt channel number (format ARG-N) causing '645 Process' termination.	
251	2	Unused	
253	1	Pointer to 635 escape coding	Set up by 645 loader
254	1	Pointer to supplement initiated termination routine.	
255	1	Pointer to normal termination routine	

Figure 5. Communication Region in 635 Slave Memory.

The termination routines, whether Supplement initiated or normal, have the responsibility for collecting information on machine conditions and writing this information, together with a core dump, onto a file (file code CR) in the same format used by the simulator in Phase 1.

Segment 'escape'

When entry point 'finish' is called, termination of the '645 Process' is effected by the normal termination routine. Segment 'escape' gets to this routine by transferring indirectly to segment 5 location 255.

```
tra  =   its(5,255,*),*
```

Escape to perform 635 escape coding is performed by saving the argument list pointer in segment 5 location 240, putting the escape "number" in index register 7, saving control in segment 5 location 242, and transferring indirectly to segment 5 location 253.

```
escape:  save
         eapbp   ap|4,*
         stpbp   =its (5,240),*
         ldx7    ap|2,*
         stcd    =its (5,242),*
         tra     =its (5,253,*),*
         return
```

Further details about the use of escape coding are available in the appropriate writeup in MSPM section BE.7