Programming Staff Note 25

Computation Center
May 27, 1964

From: J. H. Saltzer, T. N. Hastings, and R. C. Daley

Subject: Unified control of enabled user traps, including memory protection and relocation.

There are now available to a foreground user a number of "traps" which are requested and enabled by the user via a variety of supervisor subroutine calls. These traps include a high speed I/O (data channel) trap, clock trap, interrupt button trap, and the floating trap mode. The various conventions used in these enabling calls can be simply collected into a single call to a supervisor program, including provisions for specifying operation of the protection and relocation mode. The supervisor subroutine which is called would be an interpreter, and would be used typically as follows:

```
TSX   $ENABLE,4
LPI   A
LRI   B
ENB   C
AXT   **,4
TRA   **
```

The calling sequence begins in location 1,4 and ends with the TRA instruction; in between are any number of trap enabling instructions or NØP's in any order. The supervisor subroutine would enable all (permitted) requested traps and modes and interpret the AXT and TRA instruction thereby providing an automatic grace period before any traps or modes take effect. (The AXT is optional.) Following a trap of any kind which was enabled by the ENABLE sequence, all traps are inhibited; if a data channel, clock, or interrupt button trapping signal occurs it is remembered until the next ENABLE sequence, which enables that trap. Instructions permitted in the calling sequence are:

```
LPI   LFTM   ETM
LRI   EFTM   LTM
ENB          TXH   LIST,,N   (the TXH is a pointer word to
                               an enable list of the form
                               specified in CC-226.)
```

Note that a basic philosophy in the specification of the ENABLE subroutine is that the original hardware trapping procedures are imitated as closely as possible. Two new modes have been introduced with this call; their proposed operation is described below.

Two "supervisory" applications of the time-sharing system have need for some sort of core memory protection and collection of other protection mode violations; no doubt other supervisory applications could use these features. The two applications referred to here are the grading and monitoring of student class programs, and the operation of debugging programs such as FAPBUG or MADBUG. In both of these applications, an objective is for a core B supervisory program to maintain control no matter what an undebugged program happens to do by mistake. In the caseof a class an additional objective is to maintain security, say, of grades, and avoid cheating or similar malicious actions by a student.

Memory Protection Mode.

Following a call to ENABLE which specified a setting of the protection mode register, the operation of the computer as seen by the user's program would be modified: all memory protect violations including illegal instruction traps and core-A supervisor calls would cause all traps to be disabled and control to pass to the appropriate lower core location (33_8). Following a trap actual memory protect would still be in operation with memory bound returned to its earlier position and core-A supervisor subroutine calls honored normally.

Since the core-A supervisor is normally in the business of sorting out the meaning of protection traps, it would be convenient if it could do such a preliminary sorting before returning to core-B simulating a trap to the user program; a trap code could be placed in the decrement of location 32 along with the ILC at the point of violation. The following memory protection violations might be distinguished:

1. TIA to a legal supervisor subroutine.
2. Other TIA's and illegal instructions.
3. Core protection violation.
4. HTR encountered. (?)

Relocation Mode

If a call to ENABLE specified a setting of the relocation register, all addresses beginning with the address of the TRA at the end of the calling sequence are relative to the new relocation setting.

If a trap of any kind occurs while in relocation mode, the unrelocated ILC is stored, the relocation register is reset to its value before relocation mode was entered, and the trap made to the user's apparent absolute lower core location. Thus it is possible for a controlling program to get back to the interrupted program by placing the contents of location 32 in the address of the TRA at the end of the next ENABLE sequence.

Other desirable features: Although it would probably require a little more effort to add, the following feature would be very useful to a supervisory program attempting to interpret another program or permit the other program certain supervisor calls. If the TRA instruction in the calling sequence is a TIA to a supervisor subroutine name, this would signify that the supervisor subroutine should be called after IR4 is reloaded from the address of the AXT, but that the return from the supervisor subroutine should be considered to be a protection violation. (That is, a trap occurs after the subroutine has finished). The supervisor subroutine would respect the core-B user's setting of the relocation and protection registers.

When this special kind of "delayed" trap occurs, the ILC location would be filled with the relative location to return in the core-B program following the permitted supervisor subroutine call. It would be useful to add two more trap codes to the decrement of the ILC location:

5. Returning from a supervisor subroutine.
6. Error return from a supervisor subroutine.

Relation to older trap-enabling procedures.

Unfortunately, a number of traps have already been implemented in the time-sharing supervisor, with diverse techniques of specification. Here, therefore, are several suggestions for unification which will have to be reviewed very carefully in the light of the amount of reprogramming of user programs they might cause.

Clock trap.

The clock trap should be a precise imitation of the 7094 interval timer procedure. A call to supervisor subroutine CLØCØN causes location 5 to begin incrementing, but unless the clock is enabled overflow only causes a trapping signal, not a trap. This signal is remembered until the next ENABLE sequence which specifies an ENB instruction which enables the clock (bit 17 of the enable word). Supervisor subroutine CLØCØF stops further incrementing of cell 5. Thus CLØCØN and CLØCØF act only as the console on-off switch.

Console interrupt button trap.

With the above techniques in mind, the console interrupt button feature can be easily incorporated into the sequence by enabling it with an unused bit in the enable word, such as bit 18. The trap return locations would be set by a call to SETBRK, and such a call would cause all future interrupt signals to either cause a trap or be remembered, depending on whether or not an appropriate ENB instruction had been received. Since the user program has both memory protection and the ability to remember and put off interrupt button traps until it is able to handle them, the need for interrupt levels is eliminated.