Appendix J

March 5, 1965

TO: Robert Graham

FROM: J.H. Saltzer

SUBJ: Assembly Programs in a Time-Sharing System

It appears that a number of special effects caused by the time-sharing environment can be used to advantage in the design and implementation of an assembly program which is to be used exclusively on the time-sharing system. Some of these special effects should probably be carefully considered by the implementors of the 635 assembly program.

Assembly Listings

One important difference between batch-processed program debugging and the time-shared equivalent lies in the relative significance and usefulness of the "assembly listing", a printed record of the source instructions and the machine language translation, along with other items of interest, about the program or the translation. In particular, in a batch-processed system, the programmer virtually always has an up-to-date assembly listing when debugging his program. On the other hand, the characteristics of the time-sharing interaction mechanisms with turn around times measured in seconds are such that the programmer rarely has an assembly listing of the exact program he is debugging. In fact, he is likely only to have an assembly listing corresponding to the state of his program yesterday, or at best, a few hours ago, and he finds this listing of little more value than a simple listing of his source program at the same stage. We may conclude, then, that during the debugging stage of a program, any extra effort put forth by the assembly program to produce an assembly listing (except in those cases, perhaps once a day, where the programmer specifically decides to obtain off-line output) is largely wasted.

The fact that the assembly program can avoid the production of an assembly listing has been recognized in the 7094 time-sharing version of FAP, which produces an assembly listing file only

if it is requested. However, in the 7094 FAP case, only the time required to write the listing file is saved, and no important internal changes to the structure of the assembly process have been made.

It is suggested that the functions of program translation and assembly listing production can be profitably broken up into separate phases, allowing the translation process alone to proceed with much higher efficiency and less storage space. This could be accomplished by creating a "three-pass" assembly program, to use the common jargon. The first two passes would be more or less conventional assembly passes, modified so as to completely ignore the possibility of an assembly listing. The third pass, then, only done if specifically asked for, and only called into core memory in that case, would reread the symbolic input program file and the binary output program file, and collate them to produce a complete output listing file.

If the first two passes of the assembly program, then, can be divorced from the problem of creating an assembly listing, a number of advantages in speed and space can be realized:

1. There need be no coding in the main body of the assembly program which pertains to the assembly listing pseudo-operations, except that necessary to ignore them in "pass one".
2. Since no assembly listing is produced, it is not necessary to pass complete card images from pass one to pass two via a collation file; at most, about three machine words per symbolic instruction should suffice. There need only be enough of an instruction to identify the operation field and the variable field. The symbolic location field of each source instruction may be discarded during pass one of the assembly. Assembly listing pseudo operations do not need to be passed through the collation tape.
3. Thus the "collation tape" for a 1000-instruction program will probably only run to about 3000 words; this much information can probably be stored more effectively in memory than in a disk file. In fact, with the segmentation and paging scheme proposed for the 635, the assembly program should place this buffer in core memory; if the

supervisor concludes that this information really belongs on the drum memory, it can take care of the details with its paging ability.

Error Diagnostics

One can argue that the lack of a complete collation tape will make production of diagnostic information difficult; this objection can be met by smoothly tying in an editor program such as that modeled by ED on the 7094 CTSS.

Most diagnostics, such as undefined symbols, etc., in an assembly translation, are not discovered until pass two by which time the original symbolic card has been lost; the error diagnostic can include little more than the operation and variable fields of the card in question, and the relative position of the instruction from the beginning of the program. In such a case, the assembly program should immediately and smoothly turn control of the situation over to the system editor program, which can locate and print out the complete instruction in error, and allow immediate correction of the difficulty with the editing tools that the programmer is familiar with. The editor should, of course, also have facility to smoothly return control to the assembly program; depending on the organization, this might be automatic upon completion of editing.