Identification:

Overview of the Multics MAD Compiler

E.I. Ancona

Purpose:

The purpose of this document is to describe the relationship between the general specifications of the Multics MAD compiler, the MAD language and the compiled program.

Section BZ.3.01 will describe the language in detail

Section BZ.3.02 will describe the organization of the compiler and the main data bases used.

The compiler:

a. The Multics MAD compiler ~~need~~ will not be compatible with previous MAD compilers. However, programs written for the old MAD compilers should be manually convertible to Multics MAD with relative ease and the procedure for such a conversion will be documented. The incompatibility is due to the several changes made necessary by the GE-645, such as the fact that there are four characters/word rather than six. It would clearly be unwise to design a compiler on the GE-645 to translate correctly programs which used the six character/word property of the old machine. Furthermore, the addition of automatic, static and common storage classes render the old storage class declarations inapplicable. The programmer with a program written in the

old MAD must now decide in which class his variables fall. Any attempt to assign a storage class by machine would not be in the best interests of the programmer.

b. The compiler will be written in EPL and will be a standard Multics procedure. It will as far as possible conform to "Standards for Command Writers'." (BX.0.01)

The compiled program:

a. The result of the command to compile the segment a.mad will be a call to the MAD compiler, which will create the ascii file a.eplbsa, followed by a call to the eplbsa assembler which will create the segments a.text and a.link.

b. Upon assembly, the procedure segment generated will be a pure procedure and is thus recursive.

c. Standard Multics call, save and return sequences will be used for all calls to external procedures. Thus, most programs written in EPL may be called by MAD programs and ~~vice versa~~ all programs written in MAD may be called by EPL programs. MAD programs may not call EPL programs which require structure names, file names or area names as arguments since these are not implemented in MAD.

d. There will be no block structuring at all in MAD. Thus, internal functions will not use the standard call, save and return sequences. This implies that internal function names may not be passed as arguments to external procedures.

e. The returned value of a function will be considered as an additional argument of the calling program and thus the latter will provide the storage for it. (This is the EPL convention.)

The Language

a. Any statements in the old MAD which are either inapplicable in the Multics environment, or that Multics already implements, will be eliminated. In the former class, for example, is the erasable declaration, and in the latter class are the list processing statements. It should be noted that the list processing statements were useful in the old MAD for recursion. In Multics MAD, all external functions will be considered recursive. (See BZ.3.01)

b. Additions to MAD will be limited to those statements whose absence would prove a severe handicap to a programmer using MAD on Multics. Thus, only two main additions are contemplated: the capability for intersegment referencing and the option of choosing among three storage classes: static, automatic, and common. It should be emphasized that the default storage class will be automatic. (See BZ.3.01).