MEMO TO: F.J. Corbató
R.C. Daley

FROM: R.M. Graham RMG

SUBJ: Thoughts on a New Assembler with a Compiler Interface

DATE: January 22, 1969

The following thoughts represent my current thinking as to a direction for fruitful investigation of the design of a new assembler. This assembler would have a special interface for compiler generated programs between pass 1 and pass 2. This is not a completely thought out plan and hence should be taken as suggestions for further consideration rather than a firm proposal.

Pass 1 would generate three segments. A combined symbol table in the first, comments for the listing in the second, and instructions in the third. The combined symbol table would contain one entry for each literal, ordinary symbol, link, entry point definition, external definition, location counter, etc. The comment segment would contain all information which was to be printed as part of the output listing, including source statements. The format would probably be something like a character count followed by a string of characters which constitutes one line of commentary in the output listing. The instructions in the third segment would be partially assembled and in coded numeric form. Each instruction would occupy one or more words. Each instruction would contain the following items.

1. A numeric op-code

2. The index in the comment segment of any commentary which is associated with this instruction. The final output listing from the assembler would be produced by merging the instruction (in octal), its location address, and the associated commentary in the listing segment. This would result in a minimum movement of the original source program if an output listing was desired. On the other hand, if no listing was wanted, then the source text would be thrown away immediately during pass 1.

3. The modifier and the settings of bits 29 and 29.

4. The address expression or a pointer to the address expression. Depending upon the coding, if the address is simple enough (such as constant) then the actual address would appear in this field. Otherwise, it may point to a symbol table entry, if the address is a single symbol, or to a coded expression. A coded expression consists of constants and pointers to symbol table entries combined with the arithmetic operations permitted in address expressions. It might be wise to restrict the assembly language somewhat by limiting the generality of the expressions permitted in the various fields. Since it is anticipated that little actual assembly language

coding will be done and basically the assembler will be used for compilers, more limited forms of expressions would not be restrictive. In fact, expressions or even symbols might not be permitted at all in the modifier field, i.e., only numeric constants.

Also appearing in the instruction segment would be pseudo-ops which manipulate the location counters. With proper coding many instructions could probably be coded into a single word while the majority of the remaining ones could be coded into two words.

The general plan for pass 1 would then be to read the input statements, parse them and build the symbol table as it does now. In addition, if no listing was desired nothing would be entered in the comment segment. If a listing was desired, the source language statements would be transferred to the comment segment and formatted ready for merging with the instructions. In fact, they could be formatted such that the comment segment would become the listing segment by filling in blank space in the comment segment with the octal instructions and location addresses. However, this would make it more difficult for a compiler to interface with pass 2. Pseudo-ops would be processed generally, as they have been in the past. However, such pseudo-ops as USE would be translated into pseudo-ops in the instruction segment for changing the location counter value in pass 2. In addition, pass 1 would also parse and analyze the contents of the variable field of all instructions. Any expressions found there would be reduced as much as possible by using current values in the symbol table in the hope of reducing it to a constant. Expressions that could not be reduced to a constant would be transformed into coded form and put, along with numeric op-code, into the instruction segment for final translation in pass 2.

Since the input to pass 2 consists solely of the three segments, instruction, combined symbol table, and comment, any compiler which wished to use only pass 2 of the assembler would build these three segments and call pass 2 of the assembler directly. The suggestions for the format of these are, in my opinion, relatively easy for a compiler to build, since in many cases it already has the information available in a similar format.

Pass 2 of the assembler would then proceed in a fashion similar to what it does today with the principal exception that it does not parse the input statements, but instead works on the instruction segment. It would have to read instructions from the instruction segment and finish assembling those which still had expressions in the address field, interpreting the coded format.

The symbol table is not something
which a compiler is often ready to produce
A similar coded interface for pass one might
be almost as effective.