CONTROL DATA CORPORATION • DEVELOPMENT DIV • SOFTWARE DOCUMENT

DOCUMENT CLASS ______ PAGE NO i

PRODUCT NAME FORTRAN Study Project

PRODUCT NO. X010 VERSION ______ MACHINE SERIES 64/6600

DEPT NO 254 PROJECT NO 4P631 CHANGE NO ______ DATE 18 April 1966

SUBMITTED		REVIEWED				APPROVED	
[illegible]	4-18-66	[illegible]	4-18-66				
PROJ MGR	DATE	DEPT MGR	DATE	DIR	DATE	MAB	DATE
[illegible]	[illegible]						

OUTLINE OF REPORTS ON

"FEASIBILITY STUDY OF 64/6600 FORTRAN VERSION 3.0 AND CONVERSATIONAL FORTRAN"

1 2 3

FORTRAN Version 3.0 Study Report Outline

I. Current Competitive Position

A. Sales Problems and Current Customer Attitudes

1. Analysis of lost sales (Sandia, Paris, etc.) where execution of FORTRAN generated code has been determined by Marketing to be of major importance.

2. Current customer attitudes obtained from sources such as Communications in the VIM Newsletter, e. g. J.A. Archibald's letter in the February, 1965 issue.

B. Benchmark Comparisions - comparative execution times on various FORTRAN benchmark problems on a variety of existing computers. Some current candidates are the Weather Bureau OCEAN program, the French TRIDIA program, KAPL WATS 66 and MIRAGE, and four University of Utah mathematical programs. Times will be given for machines such as CDC 3600, UNIVAC 1108, IBM 7094-II, IBM STRETCH, and CDC 6600 (under Chippewa FORTRAN).

II. FORTRAN Version 3.0 Performance Potential

A. Anticipated Generated Code and Timing Comparision - illustrations of code to be generated by the new compiler for various sequences of FORTRAN statements compared with code given by Chippewa FORTRAN. The results will be run on the 6600 timing program and comparative times given. Examples will include most executed parts of some of the benchmark programs given above and estimates will be made of execution improvement.

B. Description of Optimization Techniques - exposition of the conversion of source statement sequences to unlimited register notation and of analysis of labels and control statements during the first pass of the compiler; description of methods used to obtain highly efficient code with special emphasis on array subscript computation, parameter association, and DO loop analysis.

III. Implementation Features

A. Language and Options - description of source language giving extensions proposed to the ASA FORTRAN standard base and user selectable compilation options.

B. Operating System Exploitation - compiler use and FORTRAN programmer access to operating system facilities including

III. B. Continued

program overlay and segmentation, disk random access, and mass core usage.

C. Installation Tailoring of System - statement of features to be included which will simplify installation modification of the system to satisfy its particular needs, e.g. expansion of the in-line function table.

D. New Product Potential - features of the design which allow the system to form the base for future developments of such products as a conversational compiler or optimizing assembler.

IV. Summary

A. Effectiveness as a Marketing Tool

B. Effect on Other Software - expected enhancement of Control Data software products (PERT,LP, KWIC, SORT) which are written all or in part in FORTRAN.

V. Cost and Schedule - current budget and schedule plans

Conversational FORTRAN Study Report Outline

I. Competitive Systems - external and internal characteristics of competitor's conversational compiling systems, e.g. IBM QWIKTRAN, using information obtained from trade journals, manufacturers' literature, and CDC Marketing sources.

II. Marketing Requirements - analysis by Marketing of the characteristics of a conversational FORTRAN which would help prolong the life of the 64/6600 product line.

III. 64/6600 Conversational FORTRAN - external and internal characteristics of the proposed system including diagnostic and debugging facilities, interpretive code execution, and source language compatibility with the batch compiler for FORTRAN on the 64/6600.

IV. Relation to Other 64/6600 Software - effect on and interfaces with proposed or existing software products including the MATS system and other FORTRAN compilers; use in debugging of other CDC software products written in FORTRAN.

V. Implementation Base - analysis of feasibility of using Chippewa FORTRAN or FORTRAN Version 3.0 as a basis for implementation of

V. Continued

Conversational FORTRAN contrasted with complete new design and implementation.

VI. Cost and Schedule Objectives

CONTROL DATA CORPORATION • DEVELOPMENT DIV • SOFTWARE DOCUMENT

DOCUMENT CLASS ______ PAGE NO i

PRODUCT NAME FORTRAN Study Project

PRODUCT NO. X010 VERSION ______ MACHINE SERIES 64/6600

DEPT NO 254 PROJECT NO 4P631 CHANGE NO 1 DATE 6 June 1966

SUBMITTED		REVIEWED				APPROVED	
[illegible]	6/1/66	FCL/[illegible]	6-1-66				
PROJ MGR	DATE	DEPT MGR	DATE	DIR	DATE	MAB	DATE

64/6600 FORTRAN Study Project

Final Report on 64/6600 FORTRAN Version 3.0

TABLE OF CONTENTS

INTRODUCTION

The attached report is the final output of the study authorized by the 6000 Series Product Line Plan. The section on Conversational FORTRAN has been omitted due to lack of information and no further report is planned. The benchmark examples were provided by the Marketing origanizations of the Home Office, France and Western Region whose assistance in these areas is appreciated. All code comparisons are relative to the Chippewa Operating System using the January 1966 compiler unless otherwise noted.

I. Current Competitive Position

A. Sales problems and current customer attitudes

The 6600 is not doing as well in the market place as could be hoped for. On running benchmark problems, it is embarassingly poor on compute speeds as well as on thruput. The most embarassing aspect is that the 6600 is only producing marginally better compute time than a machine costing one half as much and having less than one-third the compute capability, the UNIVAC 1108. The 6600 is not even greatly outstanding relative to the IBM 7094-X series and these machines have less than ten percent of a 6600's compute power. It is these two comparisons that are hurting us most in the market place, notably in the following instances:

1) The Air Force represents a market potential of from five to twenty-five 6000 series machines. They require essentially three times the power of a 7094-II. The 6600 is falling down in both compute speeds on FORTRAN jobs and thruput of the system.

2) AVCO requires a machine four times the power of a 7094-I and we fail there.

3) The Weather Bureau has an office which essentially wants to run one large compute bound FORTRAN program and on a price-performance basis we lose again, here to an 1108.

Certainly a more comprehensive survey of Marketing would produce more examples of a similar nature. The part that seems most awkward is that the 6600 is a machine with a huge computing capacity; the operating system and peripherals are structured for this environment, but the FORTRAN compiler is only designed for efficient handling of small, short-run-time jobs.

Another important area of concern is the attitude of our current customers. They have reason to be irritated when they pay super computer prices for a machine which isn't effectively faster than the old one they already had. The following excerpts from a letter appearing in a VIM Newsletter (March '66, P18-23) typify customer attidues:

"We are very greatly disappointed by the inability of the CDC 6600 to produce a significant improvement in computer throughput over our present system, a Philco 2000 Model 212...

"The potential of the hardware was very vividly demonstrated to us two years ago when the most crucial part of one of our most heavily used programs was hand-coded in machine language by a CDC expert and run in one seventh of the time required on our present system."

This letter goes on to describe what the user feels is needed with some remarks about what is wrong:

"We have left the most crucial item, software, until the last. This is the area of greatest need--an area where a complete redesign from the ground up is in order...The overall system should be at least 80% effective...The overall system is more like 10 or 15% effective, if that much. Our position is that this is too great a gap to close by simply modifying the present software. It needs to be redone from the design stage forward. It is unconscionable that the same software concepts should be adequate for the 6600 as were adequate for older machines...Yet this is what is being provided .

"The FORTRAN compiler produces object code in much the same way that earlier compilers did, with the same or lower degree of object code efficiency."

Then a very remarkable coincidence occurs, with no knowledge of the proposed Version 3.0, he proceeds to give a fairly accurate description of it, when he says what is needed:

"The one area in which this machine should be like its predecessors, is in the area of source language. This means, by implication, ASA Standard FORTRAN IV...The compiler should be a single compiler with two options...

one for "quick and dirty" compilations where little attention is paid to the quality of the generated object code, and one for optimum object code, which makes effective use of all the hardware advances embodied in the 6600...This may very well require the performance of an operations research problem for each DO loop..." (The above remarks appear to be based on the SIPROS system but are also applicable to the current system).

In another note appearing in a VIM Newsletter, (April '66, P5) more characteristics are coincidentally described:

"VIM members...feel CDC should consider (1) language extension, such as random access to the disk, recognition of ECS, (2) additional built-in functions, (3) more efficient object code whether by an optimizing pass or by a new compiler,..."

B. Benchmark Comparisons

Several comparisons have been made by our customers and prospects using benchmark programs. These programs are run to determine the relative performance of the machine on the selected programs. The following is an outline of the benchmark information relative to the 6600 that has been acquired during this study:

1. The most extensive benchmark comparison encountered on the basis of most machines used to run a single code, is the TRIDIA program. This program is a small matrix tridiagonalization (100 X 100) run by the French AEC in conjunction with our office there. The test results are summarized in the following table:

MACHINE	LANGUAGE	EXECUTE TIME IN SECONDS
IBM 7040	FORTRAN IV	148.
IBM 7044	FORTRAN IV	74.
CDC 3200	FORTRAN	72.41
IBM 7090	FORTRAN II	66.
CDC 3400	FORTRAN	63.
UNIVAC 1107	FORTRAN	41.21
IBM 7094-1	FORTRAN IV	31.35
GE-625	FORTRAN (?)	24.60
CDC 6600	FORTRAN/SIPROS	22.
CDC 3600	FORTRAN 63	21.41
IBM 7094-II	FORTRAN IV	16.50
IBM STRETCH	FORTRAN IV (last version)	15.58
CDC 6600	FORTRAN/CHIPPEWA	9.
UNIVAC 1108	FORTRAN (?)	7.78
IBM 360/40	Single Precision Machine Language	171.
IBM 360/65	Single Precision Machine Language	12.05
IBM STRETCH	Machine Language	9.00
IBM 360/75	Single Precision Machine Language	6.90
CDC 6600	ASCENT	1.25

2. An office within the Weather Bureau is primarily concerned with running one large code called OCEAN. On the basis of the performance of this code on the 6600 they are buying a UNIVAC 1108.

CDC	6600	- 372 seconds
UNIVAC	1108	- 420 seconds

3. The following group of codes are all of the standard scientific calculation variety as might occur frequently in an aerospace job shop operation. These tests were run by the University of Utah.

	1108	6600
Fourier Inverse and Transform	21 sec.	16.8 sec.
Matrix Inversion (50 X 50)	6 sec.	3.74 sec.
Boundary Value - Finite Difference	5 sec.	1.29 sec.
Boundary Value - Monte Carlo	36 sec.	29. 5 sec.

These comparisons are indicative of what is happening in every benchmark situation we encounter. The comparisons are not complimentary, especially with the fact that the 1108 costs only half as much as a 6600. Other comparisons run by Western Region marketing indicate that the increased speed, parallel instruction execution capability and instruction stack are not being utilized effectively; FORTRAN programs are running only 2.7 times as fast on the 6600 as on the 6400. An indication of the quality of the generated code can be obtained by comparing the FORTRAN speed of the 6600 with the 3600, whose compiler produces quite good code. In the comparison the 6600 is only 3.8 times as fast as the 3600.

II. FORTRAN Version 3.0 Performance Potential

This section of the report will give examples of code, which Version 3.0 is expected to generate, and how it goes about generating code in general. All comparisons made with Chippewa are relative to FORTRAN Version 1.0 of the compiler. The Version 3.0 code was generated by simulating the operations the compiler would go through.

A. Anticipated Cenerated Code and Timing Comparisons

Following are examples of code to be generated by the Version 3.0 FORTRAN compiler. The FORTRAN statements were taken from current benchmarks in most instances.

1. Perhaps the shortest common benchmark is the inner DO loop of a matrix multiply -

```
      DO 1  K=1,M
    1 C (I,J) = C (I,J) + A (I,K) * B ( K,J)
```

The following times are for each iteration through this DO loop for the relevant compilers:

UNIVAC 1108 FORTRAN IV	7.375 sec.
Chippewa FORTRAN	6.7 sec.
Version 3.0 (Standard)	3.7 sec.
Version 3.0 (Flash)	1.6 sec.

```
Standard Code - .D01   SA1    B1            B1 - address of A(I,K)
                       SA2    B2            B2 - address of B(K,J)
                       SA3    B3            B3 - address of C(I,J)
                       FX0    X1 * X2
                       SB1    B1 + B4       B4 - first dimension of A
                       SB2    B2 + B5       B5 - one
                       SB6    B6 + B5       B6 - loop count, negative
                       FX5    X0 + X3
                       NX6    B7   X5
                       SA6    B3
                       NE     B6 B0   .D01
```

```
Flash Code - .DO1   FX0        X1 * X2          B1 - first dimension of A
                    SB3        B3 + B2          B2 - one
                    SA1        A1 + B1          B3 - loop count
                    SA2        A2 + B2          A1 - A(I,K)
                    FX3        X6 + X0          A2 - B(K,J)
                    NX6        B5   X3          X6 - C(I,J)
                    NE         B3   BO   .DO1
```

2. The following DO loop is from the Weather Bureau's program OCEAN. This particular DO loop represents 43% of the Chippewa execute time:

```
     DO  517  I=3,IPA
     R = GG (I) * PA(I+1,J) - HH(I) * PA(I,J)
       + OO (I) * PA(I-1,J) + (PA (I,J+1)
       + PA(I,J-1)-2. * PA(I,J))/DSSQ - ZETA (I,J)
     PA (I,J) = PA(I,J)+R*HHP (I)
     IF (ABS(R).GT.ABS(CRIT3)) NN=NN+1
517  CONTINUE
```

Chippewa FORTRAN -	400 Minor cycles
Version 3.0 Standard -	162 Minor cycles

The Version 3.0 (Standard) code generated for this and following examples is contained in an appendix.

3. This is another DO loop taken from the most critical area of OCEAN; it is nested three DO loops deep and is more typical of the bulk of the DO's encountered.

```
     DO  526  K=1,KM
     UA (I,K,J) = UA (I,K,J) + SFU
526  VA (I,K,J) = VA (I,K,J) + SFV
```

Chippewa FORTRAN -	163 Minor cycles
Version 3.0 FORTRAN (Standard) -	34 Minor cycles

4. This DO loop is from another Weather Bureau code, MARK: it is from subroutine INNER where most of the time is spent:

```
    DO   200   K=1,KMAX
    VRDFR  (I,K) = VRDFR (I,K) + (RM(I,K,JP)
                 - (RM(I,K,JM) + (RMCNGS (K)))
200 VRDFT  (I,K) = (T(I,K,JP)  -  (T(I,K,JM)
                 + TCNGSV (K))) * PSIJMC
```

Chippewa FORTRAN - 335 Minor cycles
Version 3.0 FORTRAN (Standard)- 82 Minor cycles

5. This DO loop is from the same area as the previous example:

```
   DO   14  K=1,KMAX
   RADCNG = RAD (I,K,J) * FPSIJ
   TCNGSV (K) = TCNGSV (K) + RADCNG
14 T (I,K,JP) = T (I,K,JP) + RADCNG
```

Chippewa FORTRAN - 185 Minor cycles
Version 3.0 FORTRAN (Standard)- 46 Minor cycles

Example	Chippewa		Version 3.0		Ratio	
	Speed	Size	Speed	Size	Speed	Size
2	400	33.5	162	18.0	2.47	1.92
3	163	14.5	34	4.0	4. 8	3.61
4	335	29.5	82	9.75	4. 1	3.25
5	185	16.5	46	6. 0	4. 0	2.75

Speed in minor cycles
Size in words

B. Description of Optimizing Techniques

The objective of the optimizing techniques incorporated in the FORTRAN Version 3.0 design is to take advantage of the following features available in the 6000 series central processors:

1. Multiple registers (6600 and 6400)
2. Parallel instruction execution (6600)
3. The instruction stack (6600)

Roughly, the approach used is to (1) produce code for a machine with unlimited registers, (2) do a PERT analysis of this code assuming parallel instruction issue as well as execution and then (3) generate code accordingly while simulating its execution. To do exactly this would be an enormous task but with some slight simplifications (e.g., ignoring second and third order register conflicts) the procedure can be made fairly straight forward. The techniques have been developed to take advantage of the unique opportunities presented by the 6600.

Initially a statement is converted into a register free notation, R-list.

Example:	Source	A=B/C	
	R-list:	$R_1 \leftarrow B$	fetch B
		$R_2 \leftarrow C$	fetch C
		$R_3 = R_1/R_2$	form quotient
		$R_3 \rightarrow A$	store result

Source expressions are translated directly to R-list in a single scan of the expression using a modified Polish translation technique. This translation as developed, will also produce some transformations which will result in considerable economies for the 6600. Following are some examples of apparent transformations produced:

1. $A*B/C*D \longrightarrow (A*D)*(B/C)$
2. $A*(expr)/D \longrightarrow (expr)*(A/D)$
3. $A*B*C*D \longrightarrow (A*B)*(C*D)$
4. $A/B/C \longrightarrow A/(B*C)$
5. $A/5.0 \longrightarrow A*0.2$
6. $A/(7.0*11.1) \longrightarrow A*0.0129$

The effect of these conversions can most clearly be seen in example four above. Using contemporary compiling techniques it would take sixty-nine minor cycles to evaluate this expression on the 6600, while it can be accomplished in forty-nine minor cycles with the transformation.

Once the R-list for a subprogram has been produced, it is broken into sequences. A sequence begins at the beginning of a subprogram or following the end of the last sequence, it ends at the next encountered unconditional jump or active statement label (a label is active if it is referenced by other than the preceding instruction). The R-list associated with the sequence is then scanned for redundant loads, stores, and common sub-expressions, which are eliminated. From the remaining R-list, a dependency tree is formed noting which operation must precede which; this is effectively a PERT network. Using this dependency tree and the known execution times for each instruction, the latest time for beginning each operation and completing all of the operations in the minimum time can be calculated. Due to the way these times are calculated, the earlier an operation must start the higher its starting time value, so these are called priorities rather than late start times. Using the priorities and keeping track of the function units and registers, instructions are generated on the following basis:

1. Logically issuable - All of the operations necessary for generating the operands or operations which must precede this instruction have been generated (issued).
2. Machine issuable - The destination register and function unit required are available.
3. Machine executable - The required operands are available and the instruction is machine issuable .
4. Priority - explained above.

Machine executable instructions will be given precedence over instructions which are only machine issuable. The priority will be used to select from within these groups. In addition to the above optimization scheme, the following optimizations will be implemented:

1. Elimination of jumps to the next instruction.
2. Rearrangement of branch sequences to maximize the occurrence of 1.
3. Elimination of duplicate parameter lists within a subprogram.
4. Determination of whether it is more desirable to make each reference to a formal parameter by indirect addressing or address substitution.
5. Compile-time-calculation of subscript expressions as far as possible.
6. Generation of unnormalized floating point constants when they are used for subscript evaluation.

In the optimization of innermost DO loops all of the above techniques are applied and, depending upon the nature of the particular DO loop, others may be applied. The only significantly different procedure used when applying the above techniques to innermost DO loops, is that in-stack timings are used until it is discovered that the loop will not fit into the stack; in this event, the procedure of generating instructions is restarted using out of the stack timings. The remaining techniques are all pointed at reducing the size and number of instructions generated within DO loops by placing terms in B-registers and operating on them there. This is in order to improve their chances of fitting in the stack as well as to speed up their execution. The following items are contained or are performed in B-registers:

1. Index functions
2. Updating index functions
3. Loop controls
4. Loop testing
5. Base addresses plus constant addends
6. Index function increments
7. Testing increments
8. Shift counts

Several of these items and their effects are explained below.

Throughout the generated code subscript expressions are evaluated by the index function technique. The address of an element of an array is given by the following expression:

The address of A(i,j,k) = A + (i-1)+(j-1)*I-(k-1)*I*J

The above expression may be separated into a constant part (base+constant addend) and a variable part (index function) as follows: (i,j,and k integer variables, I and J constant) :

Constant part -	Base	A
	Constant addend	-1-I-I*J
Variable part -	Index function	i+j*I+k*I*J

The base plus constant addend will appear in the generated code leaving only the index function to be calculated at execute time. For referencing an array named A (dimensioned 10 by 10 by 10) with the term A(L,M,N), the instruction A-111+B1 might be generated; we assume that the index function L+M*10+N*100 has been calculated and placed in B1 If the reference occurs in a DO loop, such as follows, the compiler will precalculate the amount the index function varies for each pass through the DO and place it in a B-register before execution of the DO begins:

```
    Example:          DO  1  M=1,10
                    1 A  (L,M,N) = 5.1
```

In this case, the index function can be seen to increase by ten for each iteration. Here the index function need be completely calculated only once outside the DO and the updating done inside the loop by merely incrementing the index function by ten. The generated code could therefore be as follows:

```
    LOOP     SA1            "5.1"
             BX6            X1
             SA6            A-111+B1
             SB1            B1+10
             SB2            B2 - 1
             NE             B2  BO LOOP
```

Another optimization used in the example above was that the number of iterations through the loop was calculated before entering the loop (during compilation in this case) and placed in B2.

The next optimization is to place as many constants and addresses as possible into the remaining B registers resulting in the following code:

```
    LOOP     SA1            B3
             BX6            X1
             SA6            B4+B1
             SB1            B1+B5
             SB2            B2+B6
             NE             B2  BO LOOP
```

This code occupies one and three quarter words, a forty percent saving in space.

1. If within an adequately well behaved DO an index function occurs which is only a function of the DO variable, M in the above case, the index function will also be used as the loop counter. This requires a different definition of an index function than has been given above. This optimization will result in the freeing of another B register for other use.

2. Each sequence will be checked to see if it is only entered from one point in the program. If this is the case, and the place it is entered from physically preceeds it in the subprogram, the contents of all registers at the time it is referenced will be preserved and used as the register initial conditions at code generation time for the sequence.

3. Loads of variables which are independent of the DO's execution will be loaded prior to executing the DO when judged desirable.

4. Stores of variables whose addresses are independent of the control variable, will be stored only after the execution of the DO when judged desirable.

5. Operands appearing early in the dependency tree will be loaded before entering the loop and will also be loaded at the bottom of the loop.

Using these features the above example could result in the following code being generated:

```
LOOP    SB2           B2+B3
        SA6           A6+B1
        NE            B2  BO LOOP
```

A one word loop!

III. Implementation Features

A. Language and Options

The language planned to be implemented in the 64/6600 FORTRAN Version 3.0 compiler is a superset of ASA FORTRAN. The following criteria were used in selecting which extensions would be implemented.

1. Difficulty of implementation.
2. Effect on object code.
3. Utility to programmer.

In all cases if a user wants to use only ASA FORTRAN, a control card switch is provided to cause the printing of information diagnostics pointing out where such extensions were used.

The following extensions are planned for implementation:

1) Seven character symbols
2) DOUBLE for DOUBLE PRECISION
3) Octal constants
4) ENTRY statement
5) Less subscripts than dimensioned
6) $ statement separator
7) ENCODE/DECODE
8) O conversion for octal
9) R conversion for character information and L
10) T control for column assignment
11) Two branch logical IF
12) Mixed mode arithmetic
13) Non-standard subscripts
14) Masking statements
15) Computed GO TO followed by an arithmetic expression
16) Short DO notation with single subscripts in DATA statements
17) 3600 form of the DATA statement

Others may be added when deemed desirable. In addition statements are planned to permit the programmer to utilize other features of the operating system as soon as they are firmed up, e.g., random access to the disk.

The following options will be permitted on control cards:

1) List source program
2) List object program
3) Produce COSY
4) Produce binary deck
5) Compilation space
6) Degree of optimization
7) ASA switch
8) Cross reference listing

The ability to intermix FORTRAN and COMPASS subprograms will always be permitted.

B. Operating System Features

Overlay and Segment - Both of these features will be available to the FORTRAN user. The overlay linkage that will be provided is planned to be quite similar to that in 3600 FORTRAN. Nothing is required of the FORTRAN language in order for the user to utilize the segment system as currently planned. It is anticipated that the compiler will use the overlay system itself in order to minimize compile time and central memory requirements.

REP Control - Through the mutual efforts of the FORTRAN, COMPASS and SCOPE projects a new pseudo-op has been developed which will generate cards for the loader to use in filling up large areas preset by DATA statements. This was done to minimize the size of binary decks. Consider the case where a large array is set to contain all zeros by a DATA statement. Previously this would require a binary card for every one to thirty locations so filled. With the REP control the loader is told that a given number of data words are to be repeated N times at increments of M words. In effect, the DATA statement is executed at load time.

Mass Storage and Mass Core - As stated earlier, statements will be provided to permit random access to files on the disk. The following statement is now under consideration:

READ (FNME (K), N)

Where:

FNME	is the name of the file
K	is the record within the file
N	is an optional FORMAT statement number

Mass core is to be handled in a rather singular fashion. Files are to be declared in a new declarative statement. The placement of variables within the files will be similar to placing variables in COMMON blocks.

Example: ECS/A14/SKSK (500), C,X (1000,1000)

No single dimension of an array held in mass core may exceed $2^{17}-1$, however, the total array size may be up to $2^{21}-1$. An element in ECS may be referenced via block transfers between ECS and central memory. These will be performed by a new statement, READ/WRITE ECS. The only other way these elements may be used is as parameters in calling sequences.

The compiler hopes to use ECS for storage of overlays and R-list between phase of compilation and/or as a buffer for generated code if possible.

C. Features for Use by the Installation

The compiler is designed to use macros for all in-line functions such as type conversion. This mechanism is set up to be open ended to permit each installation to add to the list of in-line functions in a very straight forward fashion. An example of such use might be the incorporation of SQRT as an in-line function to produce faster, although bigger SQRT calculations.

A feature currently under consideration is the incorporation of a standard FORMAT statement to be specified by each installation. This could be referenced as FORMAT number zero. Using this might facilitate non-programmer types in both writing their programs and preparing data cards.

D. New Product Potential

One of the major features of this design is its modularity in overall design as well as internal structure. This is brought out by the following list of overlay characteristics.

Pass 1
Phase 1 -
- a) Generates all code associated with declared variables storage allocation.
- b) Outputs COMPASS source lines
- c) Handles an entire BLOCK DATA subprogram without the need of other overlays.

Pass 1
Phase 2 -
- a) Allocates space for usage declared variables
- b) Converts FORTRAN source lines to register free notation, R-list.
- c) Generates macros, in R-list notation, for ASF's.

Pass 2 -
- a) Expands macros
- b) Eliminates redundant code
- c) Performs PERT-like analysis of code
- d) Outputs COMPASS lines

By using these large "building blocks" other new systems for the 6000 series become economically and technically more feasible; in particular the following:

Optimizing Assembler - COMPASS source lines could be converted to the register free notation, R-list, by a translator. Pass 2 could read in the R-list and optimize it as it currently does the FORTRAN generated R-list. The shuffled code could then be passed on to COMPASS for normal assembly. Indications are that this type of optimization performed on hand generated assembly code is capable of producing speed improvements in excess of 50%. Further, this relieves the programmer from having to learn the timing characterisitcs of the machine and still produce good code.

Conversational FORTRAN - Using phases one and two souce lines can be cracked and converted to R-list. The R-list could be used as the object language and executed interpretively. The COMPASS lines produced would provide for the storage of constants and variables.

IV. Cost and Schedule

A. Cost and Schedule

This project is expected to cost $315,000. The following dates are based on a June 15, 1966 go-ahead on the project:

	Current Objective	Plan Objective
SIR approved	6/15/66	
DO submitted	6/15/66	
GED submitted	7/5/66	5/66
ERS submitted	8/16/66	9/66
IRS completed	8/23/66	
Demonstration Date	4/11/67	1Q/67
System Checkout completed	5/2/67	
Release to field	6/15/67	2Q/67

B. Schedule Dependencies

As mentioned above the schedule depends on a go ahead date of 6/15/66; an earlier date would result in an equally earlier release. The major problem anticipated in producing this system will be adequate manpower, both in numbers and ability. The study project has been trying to get one man for two months with no success to date. If similar lags are encountered from the go-ahead date on it will have a commensurate effect on the release date of the system (Perhaps this problem can be minimized by the hiring of contract personnel).

V. Summary

A. Effectiveness as a Current Marketing Tool

"If to-day Control Data could offer an operational FORTRAN just twice as fast as Chippewa FORTRAN, and nothing else but plans, we would have what we really need to-day, and we would be trusted for the plans we have. I know that there is no use to look backward; however, I think that the same statement would be true a year or two from now. In other words, I still believe that the problem No. 1 in the software situation for 6000 series is FORTRAN". (Extracted from a letter to Home Office Marketing from a sales manager).

The preceding sections of the report demonstrate a software system capable of increasing the apparent compute power of the 6600 by better than 250%. The desirability of having such a product is unquestionable. The only questions remaining are 1) when can we get it, and 2) how much does it cost.

It is the judgment of the project that the system can be turned out in twelve calendar months from the time that the final go-ahead is given. This statement does not provide us with a means of running benchmarks at reasonable speeds tomorrow. It does give us an out. A salesman can say that a much better system than is currently available is on the way. The techniques have already been worked out sufficiently to convince a sophisticated customer that the mechanisms are workable and the estimated performance improvements are realistic. A promotional document can be provided for sales people to present to such customers.

With regard to the cost of the compiler, anything which would improve the performance of a system to the extent indicated for less than a million dollars would have to be considered a bargain. In this instance, the compiler is estimated to cost $315,000.

B. Effect on Other Control Data Products

There is over $400,000 worth of software currently planned to be written by Control Data or its subcontractors in FORTRAN for the 6000 series. There is probably another investment in excess of a half million dollars in software already written in FORTRAN which is intended to be transferred intact for use on the 6000 series. By providing a significantly improved FORTRAN system, the company would receive significantly superior products at no direct change in their costs.

APPENDIX "A"

Producing Version 3.0 Using Chippewa as a Base

The most efficient means of incorporating the optimizations described in the body of this report would require at least the two following operations:

1. Modify the translator to produce R-list rather than object code.
2. Write the Pass 2 processors.

The statement processors represent between 60 and 80 percent of the existing code. These would all have to be modified (rewritten is probably more accurate). The statement scanner may be retained and the assembler facilities separated out. The one pass characteristic would have to be eliminated in order to permit better understanding of the DO structure and sequence length.

The second pass processor would be written regardless.

The resultant product would probably be slowed down in delivery time by requiring the project to learn what already exists. Judging from the current status of Chippewa FORTRAN, we would be hard pressed to bring it up to the maintenance and documentation standards now prevalent in Applications products.

Summarily, Version 3.0 could be produced using Chippewa as a base. The savings resulting from this approach would be marginal or negative. The question is the same as can a Ferrari be made out of a Falcon, can a 1604 be modified to work like a 6600? Yes, but the cost of doing it is uneconomically high, and the result not as reliable as starting from the ground up.

APPENDIX "B"

64/6600 FORTRAN Version 3.0 Standard Generated Code

The following code was generated by hand by simulating the processes which will be performed by Version 3.0. The examples are described in Section II A of this report. The address term "+CA" should be read "plus constant addend".

Source Code

```
   DO  517  I=3, IPA
   R = GG (I) * PA(I+1,J) - HH(I) * PA (I,J) + 00 (I) * PA(I-1,J) + (PA(I,J+1)+
       PA (I,J-1) -2. *PA(I,J))/D SSQ - ZETAA (I,J)
   PA  (I,J) = PA (I,J) + R * HHP  (I)
   IF  (ABSCR . GT. ABS  (CRIT 3)) NN=NN+1
517 CONTINUE
```

Register Assignments:

B1	(I,J) index function	B4	shift count
B2	(I) index function	B5	scratch
B3	loop limit	B6	(I) increment
		B7	(I,J) increment

Standard Code

```
Loop    SA 1  GG-1+B2        SA 1  PA + (CA)+B1     AX2    B4     X6
        SA 2  PA+(CA)+B1     SA 2  PA+(CA)+B1       SX5    B6
        SA 3  HH-1+B2        NX 3  B5  X5           SA6    R
        SA 4 PA+(CA)+B1      SA5   ZETAA +(CA)+B1   FX0    X0 + X4
        SA 5  "2.0"          FX 1  X3 + X1          AX4    B4   X1
        FX0  X1 * X2         NX 3  B5   X1          BX2    X6 + X2
        FX1  X3 * X4         SA 1  CRIT 3           IX6    X5 + X3
        SA2  00-1+B2         FX 2  X3 + X2          BX3    X4 + X1
        SA3  PA +(CA)+B1     NX3   B5   X2          NX7    B5   X0
        FX5  X5 * X4         SA2   HHP-1+B2         FX0    X3 - X2
        FX0  X0 - X1         FX0   X3 - X0          NOP
        SA1  DSSQ            NX3   B5   X0          SA7    PA+(CA)+B1
        FX2  X2 * X3         SB4   59               PL     X0    GL1.
        NX3  B5   X0         FX0   X3 - X5          SA6    NN
        FX0  X5/X1           NX6   B5   X0          NOP
        FX5  X2 + X3         SA3   NN               NOP
                             FX0   X6 * X2          SB2    B2 + B6
                                                    SB1    B1 + B7
                                                    GE     B2 B3 Loop
```

Source Code

```
    DO   526    K=1,KM
    UA (I,K,J) = UA (I,K,J) +SFU
526 VA (I,K,J) = VA (I,K,J) +SFV
```

Register Assignments

B1 - (I,K,J)	B4 - UA+CA
B2 - Loop count	B5 - VA+CA
B3 - one	B6 - (I,K,J)Inc
	B7 - Scratch

Standard Code

```
LOOP    SA1     B4+B1       FX1     X3+X4
        SA2     SFU         NX7     B7  X1
        SA3     B5+B1       SA6     B4+B1
        SA4     SFV         SA7     B5+B1
        FX0     X1+X2       SB1     B1+B6
        SB2     B2+B1       NE      B0 B2 LOOP
        NX6     B7  X0
```

Source Code

```
      DO   200  K = 1, KMAX
      VRDFR (I,K)=VRDFT (I,K) + (RM(I,K,JP) - (RM(I,K,JM) + RMCNGS (K)))
200   VRDFT (I,K)=(T(I,K,JP) - (T(I,K,JM) + TCNGSV (K)))*PSIJMC
```

Register Assignment:

B1	(I,K,JM) index function	B4	(I,K,JP) index function
B2	K	B5	scratch
B3	loop limit	B6	(I,K) index function
		B7	415

Standard Code

```
    SA1  T-4566 + B1               FX0          X5 - X3
    SA2  TCNGSV -1 + B2            SA3          VRDFT - 82 +B6
    SA3  RM-4566 + B1              NX5          B5   X0
    SA4  RMCNGS - 1 + B2           FX6          X2 * X1
    FX0  X1 + X2                   FX1          X5 + X3
    SA5  RM - 4566                 SB1          B1 + B7
    NX1  B5   X0                   SB4          B4 + B7
    FX0  X3 + X4                   NX7          B5   X1
    SA2  T-4566 + B4               SA6          VRDFT - 82 +B6
    NX3  B5   X0                   NOP
    FX7  X2 - X1                   SA7          VRDFR - 82 +B6
    SA1  PSIJMC                    SB2          B2 + 1
    NX2  B5 X 4                    SB6          B6 + 81
                                   GE           B3 B2 Loop
```

Source Code

```
   DO     14  K = 1, KMAX
   RADCNG = RAD (I,K,J) * RPSPIJ
   TCNGSV (K) = TCNGSV (K) + RADCNG
14 T (I,K,JP) = T(I,K,JP) + RADCNG
```

Register Assignments:

B1	(I,K,J) index function	B4	(I,K,JP)
B2	KMAX	B5	Scratch
B3	K	B6	415
		B7	T - 456

Standard Code

```
LOOP    SA1  RAD + B1           SA6       RADCNG
        SA2  FPSPIJ             FX5       X6 + X4
        SA3  TCNGSV + B3        NX6       B5   X5
        SA4  B7 + B4            SA7       TCNGSV + B4
        FX6  X1 * X2            SB3       B3 + 1
        SB1  B1 + B6            SA6       B7 + B4
        FX0  X6 + X3            SB4       B4 + B6
        NX7  B4    X0           GE        B2 B3 LOOP
        NOP
```

CONTROL DATA CORPORATION • DEVELOPMENT DIV • SOFTWARE DOCUMENT

DOCUMENT CLASS	SIR
PAGE NO	i
PRODUCT NAME	64/6600 FORTRAN
PRODUCT NO.	C010
VERSION	3.0
MACHINE SERIES	64/6600
DEPT NO	254
PROJECT NO	4P6X1
CHANGE NO	
DATE	

SUBMITTED

PROJ MGR	DATE
JFT	6/14/66
[illegible]	6-14-66

REVIEWED

DEPT MGR	DATE	DIR	DATE
[illegible]	6-14-66	[illegible]	6/14/66

APPROVED

MAB	DATE
[illegible]	6/14/66

SPM	DATE
R C Guenther	6/14/66

COMPANY PRIVATE

CA 138 REV 3—66

I. Product Identification

Product Name:	64/6600 FORTRAN Version 3.0
Project Number:	C010
Product Number:	4P6X1

COMPANY PRIVATE

II. Products to be Developed

A. Software: A FORTRAN compiler and necessary object time routines which will operate under 64/6600 SCOPE Version 3.0. The language will contain ASA FORTRAN as a subset and will include most of the features of 64/6600 FORTRAN Version 2.0.

B. Customer Documentation and Promotional Material: GIM, Instant, Reference Manual, series of STM's, and other documents as determined by Dept. 241.

C. Design Documents: Design Objectives, General External Design, External Reference Specifications, Internal Reference Specifications.

III. Design Requirements

A. Justification
A need exists for a compiler which processes all ASA FORTRAN programs, compiles in 32K of central memory, exploits the characteristics of a 6400 or 6600 during compilation and program execution, and is easily modifiable to accomodate various installation and programmer requirements. This product should extend the life of the 64/6600 hardware by providing all customers with improved performance (see III-B).

B. External Design Objectives
The language will be ASA FORTRAN with certain extensions so that nearly all programs written in 64/6600 FORTRAN Version 2.0 or FORTRAN IV will be processed correctly without modification. Random access to records on MASS STORAGE will be provided with minor extensions to the FORTRAN language. The programmer will be allowed, in FORTRAN, to obtain full use of the program segmentation capabilities of the operating system.

B. (continued)

An optional ASA mode of compilation will cause non-ASA usages to be flagged with informative diagnostics. Two degrees of optimization of object code will be available in the initial release and facilities for incorporating a third will be provided for in the design; the first will give faster compilation but slowest execution; the second will have a high degree of analysis of DO-loops and array references, reordering of instruction sequences based on functional unit and register availability and will produce the most compact code of the three; the third would provide even more extended analysis and instruction reordering and would produce the fastest executions, but would take longer for compilation and produce slightly larger object code than the second. The object code produced for the 6600 by the second degree of optimization for compute bound problems will run at least two and a half times as fast as that generated by 64/6600 FORTRAN Version 2.0. The speed increase will be demonstrated using the crucial benchmarks provided by Marketing for the study report. The compiler will produce three classes of diagnostics: fatal (no object code produced), fatal to immediate execution (object code produced), or informative (allowing immediate execution). Object time routines will be revised to produce more diagnostics.

C. Internal Design Objectives: The compiler will be structured to operate on a 6400 or 6600 computer with 32K of central core memory. It will be designed to be loaded in overlays to allow maximum space for tables. New algorithms for table management will give very fast table search and entry. If it is allowed by the operating system, ECS, when present, will be used for segment loading and for scratch areas. The assembly phase of compilation will be done by the new assembler. Compilation speeds are expected to be fast, especially if the minimal degree of optimization is selected. Primary emphasis, however, will be placed on the generation of highly efficient object code. The system will be implemented to provide easy modification by users.

COMPANY PRIVATE

IV. Hardware Configuration

The minimum configuration allowable is a 32K 64/6600 machine with the minimum peripheral configuration required by 64/6600 SCOPE Version 3.0.

IV. Continued

As much central memory as is ~~available~~ requested will be used for compiler tables; ECS, if available, will be used for segment loading and scratch areas.

V. Software Relationships and Interdependencies

The compiler will be developed using 64/6600 ~~ASCENT~~ COMPASS Version ~~4~~1.0, and will ~~64/6600 FORTRAN Version 1.1,~~ and 64/6600 SCOPE Version ~~2~~3.0, but will be released under 64/6600 COMPASS Version 1.0 and 64/6600 SCOPE Version 3.0.

VI. Standards

The compiler will process all properly formed ASA standard FORTRAN programs.

VII. Other Actions Required

None

VIII. Recommended Responsibilities

A. Design and Implementation - Department 254

B. Recommended Design Review Board

P.P. Chavy, Chairman
S. Elkin
J.P. Kintz
J.R. Hanson
R.G. Harteker
R S Ritz

COMPANY PRIVATE

IX. Cost Objectives

A. Development, Documentation, and QA Costs.

Fiscal Year by Quarters	Development	Documentation	QA	Total
1 Q 67	$41,000	$3,600	$	$44,600
2 Q 67	62,000	6,000		68,000
3 Q 67	67,000	9,100		76,100
4 Q 67	68,000	9,300	8,000	85,300
1 Q 68	37,000	--	9,000	46,000
TOTALS	$275,000	$28,000	$17,000	$320,000

X. Schedule Objectives

	SO/CD	Present Estimate	6000L Product Plan Date *
SIR approved		6/14/66	
DO submitted	CD	6/20/66	
GED submitted	SO	7/05/66	
ERS submitted	SO	9/05/66	
IRS completed	SO	11/01/66	
Product Demonstration	SO	4/11/67	
QA Test Cases to Project	SO	4/10/67	
Product submitted to QA	SO	6/02/67	
Product Released to field	SO	8/11/67	
GIM published	SO	10/16/67	
Reference Manual published	SO	4/17/67	

* The March Plan proposed a SCHEDULE OBJECTIVE for Product Release in 2nd. Q/67. However, in the March Plan this product was relegated to further study with attendant potential extension of release date.

COMPANY PRIVATE

CONTROL DATA CORPORATION • DEVELOPMENT DIV • SOFTWARE DOCUMENT

DOCUMENT CLASS	DO			PAGE NO	i
PRODUCT NAME	64/6600 FORTRAN				
PRODUCT NO.	C010	VERSION	3.0	MACHINE SERIES	64/6600
DEPT NO	254	PROJECT NO	4P6X1	CHANGE NO	
DATE	20 June 1966				

SUBMITTED		REVIEWED				APPROVED	
[illegible]	6/20/66	[illegible]	6-20-66	[illegible]	6/20/66	[illegible]	6/28/66
PROJ. MGR	DATE	DEPT MGR	DATE	DIR	DATE	MAB	DATE

[illegible] 6-20-66

for [illegible] 6/20/66

QA Manager Date

[illegible] for [illegible] 6/20/66

DOC. Manager Date

RC [illegible] 6/20/66

SPM Date

RECEIVED
JUN 20 1966
DOCUMENT CONTROL

COMPANY PRIVATE

DISCLAIMER: THIS DOCUMENT IS A WORKING PAPER ONLY, AND DOES NOT NECESSARILY REPRESENT ANY OFFICIAL INTENT ON THE PART OF CONTROL DATA.

1.0 General Objectives

1.1 General:

64/6600 FORTRAN Version 3.0 is a compiler designed to process an extension of ASA FORTRAN generating highly efficient object code exploiting the characteristics of the machine and accomodating a variety of users with different requirements of compile time, code size, and execute time. The system operates in a 32K 64/6600 but will take advantage of any central memory or ECS space it is allocated. The compiler structure and implementation will allow easy modification by users.

1.2 Marketing Requirements:

Marketing requires a compiler which processes ASA FORTRAN, shows the hardware superiority of the 64/6600 machines, and which is easily modifiable or has selectable features to satisfy differing user requirements.

1.3 Competition:

The major competition at this time is the UNIVAC 1108; the top of the IBM 360 line will also be competitors. The UNIVAC 1108 is in a strong position as was shown in the "64/6600 FORTRAN Study Report; Final Report on 64/6600 FORTRAN Version 3.0".

1.4 Prime Objectives:

The primary objectives of this product in order of importance are the following:

1. Improvement in the execution speed of generated code by a factor of 2.5 over FORTRAN Version 2.0
2. Compatibility with 64/6600 FORTRAN Version 2.0
3. Modularity and maintainability
4. Fast compilation

2.0 External Objectives

2.1 Function Objectives:

2.1.1 General

This compiler will process correctly all properly formed

2.1.1 General (Continued)

ASA FORTRAN programs. Extensions will be implemented to increase the power of the language and to simplify the transfer of 64/6600 programs to the new system. The compiler will be capable of producing very efficient object code.

2.1.2 Advantages:

Access to mass storage will be provided by extensions to the FORTRAN language. The programmer may elect which of various features of the system he desires to use, including either of two degrees of object code optimization. He will have full access to the program segmentation and overlay features contained in the operating system. The system will be easily tailored to an installation's requirements.

2.1.3 Function Description:

Extensions to ASA of ENCODE/DECODE, PRINT, PUNCH and other features will minimize compatibility problems in transfering programs written in 64/6600 FORTRAN Version 2.0 to 64/6600 FORTRAN Version 3.0. Multiple entry points to subroutines will be implemented. Common blocks will be assignable to ECS if this capability is provided by the operating system. Routines will be provided to form the interface between the FORTRAN program and the operating system to allow full use of the segmentation features of the loader. An optional ASA compilation mode will cause the generation of an informative diagnostic whenever a non-ASA usage is recognized. The programmer may select several alternatives in the code to be generated. Two degrees of code optimization are allowed to enable him to balance execution space, execution speed, and compilation speed. A third and higher degree of optimization is feasible within this design and facilities for its incorporation at a later time will be provided in the design and implementation. A design note giving the optimization alternatives is given in Appendix A, along with some samples of the code

2.1.3 (Continued)

to be generated by the various degrees of optimization. Extensive DO-loop analysis, efficient arithmetic and logical expression handling, restriction of the domain of applicability of index functions, and delineation of flow blocks are the basis for formation of the input to the optimizer. The programmer may select a traceback feature for error tracing.

2.1.4 Language:

The compiler will initially be written using 64/6600 FORTRAN 2.0 and 64/6600 ASCENT Version 2.0 but will be distributed in 64/6600 COMPASS Version 1.0 form.

2.1.5 Distribution:

The system will be distributed in the standard form for system updating.

Hardware Interface:

2.2.1 Minimum machine configuration:

The minimum configuration allowable is a 32K 6400 or 6600 with the minimum peripheral configuration required by 64/6600 SCOPE Version 3.0.

2.2.2 Optimum Machine Configuration:

The minimum required by SCOPE Version 3.0 will be sufficient for object code (production). The addition of ECS will result in faster compilation.

2.2.3 Usage of Special or Alternative Devices:

If the operating system allows it, the compiler will use ECS for scratch and for segment loading. The compiled program will be able to randomly access records in files on disk if the operating system allows. Program accessible COMMON areas in ECS will be provided in FORTRAN.

2.2.4 Special Hardware Modification Requests:

We are investigating 1) the addition of normalization to the floating point add and subtract instructions producing a significant effect on speed and/or compactness of the object code and 2) the elimination of reservation on B0 would free another B register if 1) is not implemented.

2.3 Software Interface:

2.3.1 Operating System:

The compiler will operate under 64/6600 SCOPE Version 3.0.

2.3.2 Necessary Systems:

64/6600 ASCENT Version 2.0 and 64/6600 SCOPE Version 2.0 will be required by September, 1966 for compiler development. SCOPE Version 3.0 with File Manager and 64/6600 COMPASS will be required by December of 1966.

2.3.3 Additional Systems:

None

2.3.4 Additional Requirements:

Any program written in mixed FORTRAN and ASCENT which requires different calling sequences may require modification.

2.4 Installation Objectives:

2.4.1 Operator Communication:

In addition to the normal diagnostic messages provided by SCOPE and File Manager, some diagnostic and information messages may be displayed on the console.

2.4.2 Configuration Variance:

The compiler will be written with the possible introduction of ECS in mind. Other variances are handled by the operating system.

2.4.3 Installation Variance:

The compiler will be structured for ease of installation tailoring of the compiling system. A straight forward means of declaring and causing a function to be compiled in-line will be included. The compiler will be highly modular and the interfaces will be completely specified to aid in field modification.

2.5 Performance Objectives:

2.5.1 General:

The compiler is expected to be able to process about 10,000 cards per minute. Some examples of object code execution times are given in Appendix A. The overall speed improvement in object code should be about 2.5 times as fast as 64/6600 FORTRAN Version 2.0.

2.5.2 Dependency:

The compilation speeds are dependent on the speed of COMPASS Version 1.0 and on which mode of code optimization is selected. The presence of ECS will increase compilation speed.

2.5.3 Example:

See Appendix A for object code examples.

2.6 Limitations:

2.6.1 Hardware:

No known hardware limitations

2.6.2 Software:

No known limitations imposed by other systems.

2.6.3 Conversion:

The user currently writing in ASA FORTRAN will only be required to precede his deck with the appropriate control cards. Nearly all programs written in 64/6600 FORTRAN Version 2.0 and most written in IBM FORTRAN IV will be processed correctly

2.6.3 (Continued)

without modification. In any event, the conversion problems should be simple.

3.0 Internal Objectives

3.1 Structure:

The compiler will be constructed to operate in two passes, the first being divided into two phases. Phase 1 processes declaratives and generates assembly code. Phase 2 processes all other FORTRAN statements. Output from phase 2 consists of register free execution strings (R-list) which point to various table entries. Pass 2 performs register assignment, in line expansion of functions, code optimization and instruction sequencing, and generates assembly code in the form required by 64/6600 COMPASS. Phase 2 of Pass 1 will overlay Phase 1 of Pass 1; Pass 2 will overlay Pass 1.

3.2 Implementation:

Since the compiler must operate in 32K of central memory, it will be loaded in overlays (from ECS if possible) to allow maximum space for tables. The symbol table will be linked in a way analogous to a tournament sort technique with very fast search and entry.

3.3 Size:

The objective is to compile a sizeable program on a 32K machine.

3.4 Maintenance:

The system will use any update or conditional assembly features available as standard software.

4.0 Release Objectives

4.1 Initial Release:

All language features will be available. Only the first two degrees of object code optimization are currently planned for this product.

4.2 Subsequent Release:

No subsequent release is currently planned, however, consideration should be given to a later release which would include the third degree of object code optimization and extensions for ECS.

5.0 Applicable Standards

5.1 National:

ASA Standard FORTRAN

5.2 Applications:

5.2.1 Approved:

None

5.2.2 Pending:

None known

5.3 Proposed:

None known

APPENDIX A

Most of the code optimization takes place in Pass 2. The exception of this is that in Pass 1 some arithmetic expressions are restructured to permit parallel execution of two multiply instructions. Following are two design notes which describe the characteristics of the optimization passes available. References to APLIST, SUBLIST and unnormalized constants all pertain to subroutine linkage and an understanding of them is not required to obtain a general understanding of Pass 2.

PASS 2

Exclusive of reformation, all optimization is currently planned to take place in pass 2. In order to satisfy the requirements of the largest possible number of users, it is felt that three varieties of pass 2 will be necessary and sufficient. The following list will summarize the characteristics of each of these.

1. Produce code as quickly as possible with no extra time being spent in an attempt to generate either short or fast object code. This system is referred to as FLUSH.
2. Produce reasonably efficient code via elimination of redundant code, provision of temporary indexing in innermost DO loops and timing analysis. This system is referred to as STD.
3. Attempt to produce code which will execute as quickly as possible with little regard for the amount of time required for the compilation process. This system is referred to as FLASH.

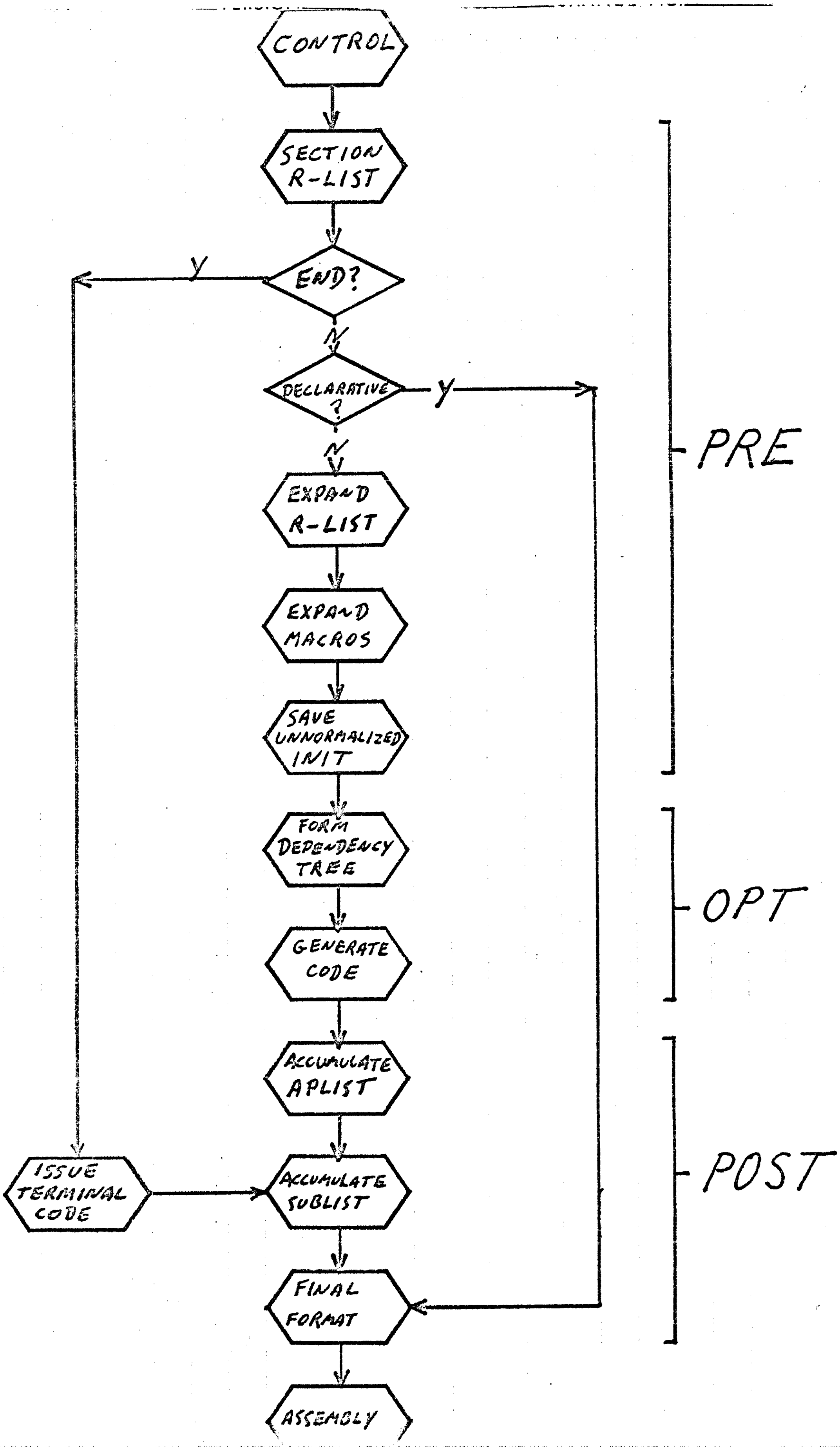
In developing these systems, it is planned that a good deal of the code will be common to more than one of them, as will be seen in later descriptions. Because of this, some optimization may unavoidably appear in FLUSH.

FLUSH

Since FLUSH is intended to be quite fast, its organization will be very straight forward. Figure 1 indicates a first cut at a general flow chart.

CONTROL reads in R-list in a continuous fashion. R list is broken into sections and checked for containing either declarative code or an end of subprogram flag. If neither of these occur, the R-list is expanded to two words per entry and all macros are expanded. Any code for initialization of unnormalized constants is retained and the remainder of R-list is passed onto OPT. OPT forms a dependency tree from the R-list. It then assigns registers and generates code through a simulation of the registers generating any temporaries required by register overflow. The output of OPT is scanned by POST noting where address substitution is to take place; this information is accumulated in SUBLIST. Calling sequences are also accumulated. The generated code is converted to an assembly line image and passed on to the assembler. When the end statement is encountered, the unnormalized constant initialization code, calling sequences (APLIST) and SUBLIST are issued in sequence for formatting and passed on to the assembler.

FLUSH

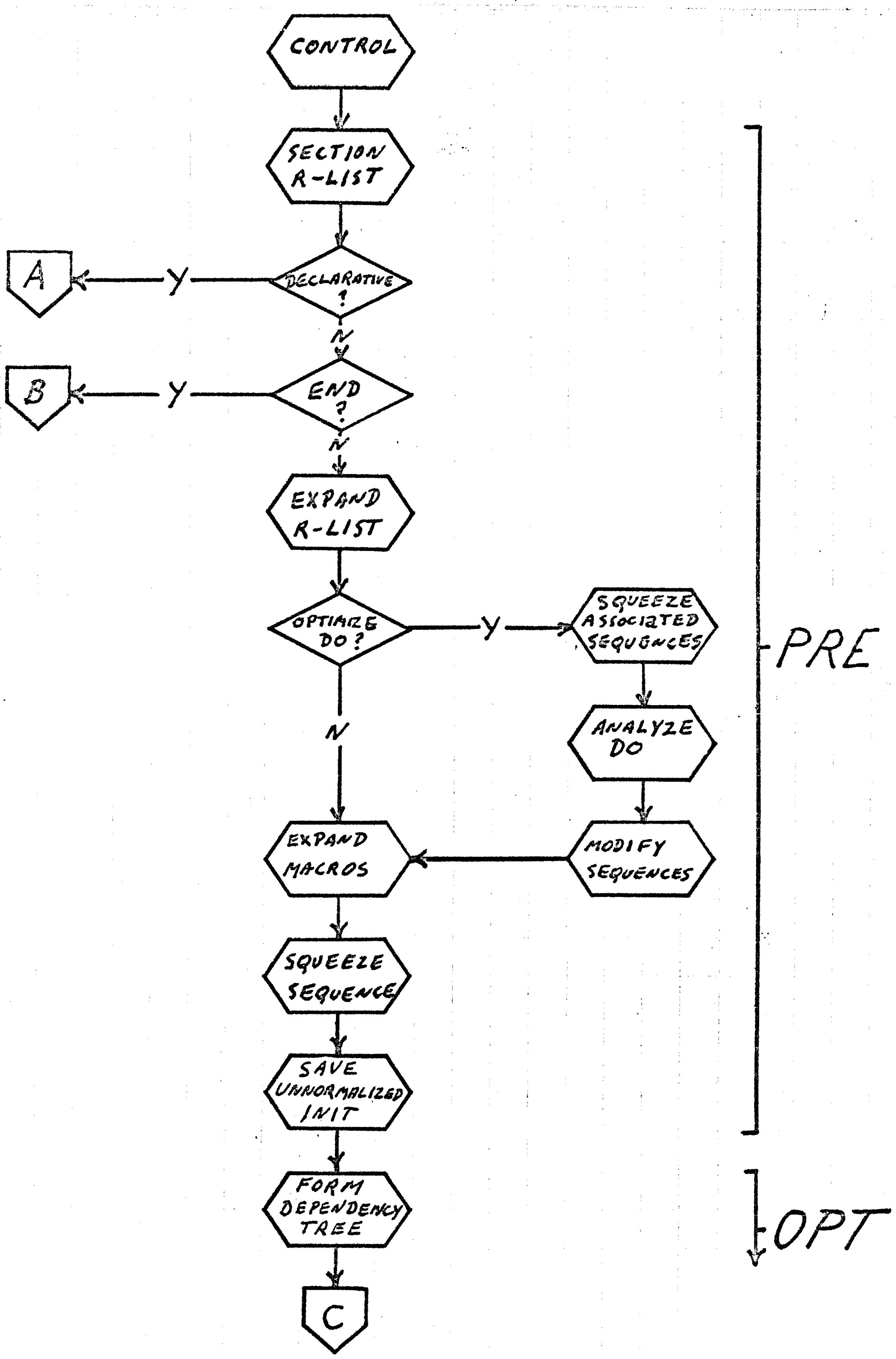

CONTROL
SECTION R-LIST
END?
Y
N
DECLARATIVE ?
Y
N
EXPAND R-LIST
EXPAND MACROS
SAVE UNNORMALIZED INIT
FORM DEPENDENCY TREE
GENERATE CODE
ACCUMULATE APLIST
ISSUE TERMINAL CODE
ACCUMULATE SUBLIST
FINAL FORMAT
ASSEMBLY
PRE
OPT
POST

STD

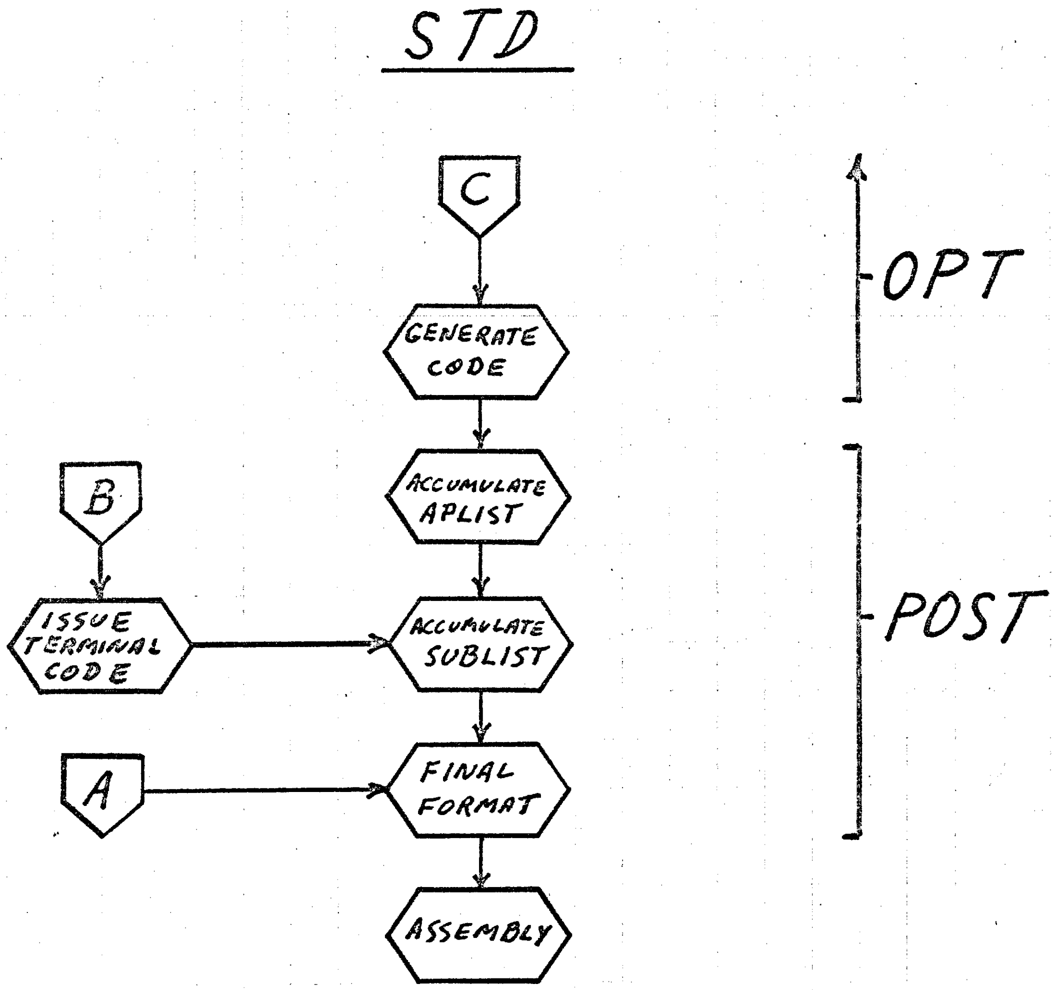
This version of PASS 2 is intended to be the one most frequently used. It will attempt to produce object code substantially superior to any other currently available FORTRAN compiler on the 6000 series of machines while maintaining reasonable compiling speeds and taking care to minimize the amount of core required. Optimization techniques utilized will include register and function unit simulation, temporary indexing of inner DO loops and elimination of redundant code.

Through the expansion of R-list, STD is identical to FLUSH. Then a check is made to determine whether or not we have encountered an optimizable DO; if we have, all of the associated R-lists are read in and redundant code squeezed out. The DO is then analyzed and R-list modified accordingly. Remaining macros are then expanded and the resultant code squeezed. From here we proceed similarly to FLUSH with the exception that the code for initialization of unnormalized constants is optimized.

STD


CONTROL
SECTION R-LIST
DECLARATIVE ?
Y
A
N
END ?
Y
B
N
EXPAND R-LIST
OPTIMIZE DO ?
Y
SQUEEZE ASSOCIATED SEQUENCES
ANALYZE DO
MODIFY SEQUENCES
N
EXPAND MACROS
SQUEEZE SEQUENCE
SAVE UNNORMALIZED INIT
FORM DEPENDENCY TREE
C
PRE
OPT

STD


STD
C
GENERATE CODE
ACCUMULATE APLIST
B
ISSUE TERMINAL CODE
ACCUMULATE SUBLIST
A
FINAL FORMAT
ASSEMBLY
OPT
POST

FLASH

The objective of FLASH is to produce the fastest object code possible generally ignoring such desirable characteristics of a compiler such as speed and space requirements. The resultant code we hope to be competitive with that written by hand, at least for innermost DO loops. Coding techniques used may include the following:

1. Those used in STD
2. Performing loads at the bottom of the DO
3. Maintenance of global variables and constants in X registers for the duration of a DO
4. Removal of code associated with expressions which are invariant with regards to the DO variable from the loop
5. Evaluation of indexed variable addresses in A registers
6. Retention of register contents when performing a branch within the subprogram.
7. Preprocessing loads and post processing stores of variables independent of the DO variable

No flow chart of FLASH follows as its structure is not adequately defined.

PASS 2 PERFORMANCE OPTIONS

The table at the bottom shows the anticipated results of compiling, under the various types of pass 2 optimizing approaches, the following FORTRAN statements;

```
      DO 1  K  =  1,N
    1 C(I,J)   =  C(I,J) + A(I,K)*B(K,J)
```

The code is for the execution portion of the loop and does not include initialization. There are two versions of STD; the entries associated with STD are for code which does not permit storage of constants in B's, whereas STDA does. It is anticipated that STDA will be the one adopted.

	FLUSH	STD	STDA	FLASH
Instruction Words	17_{10}	5	3	2
Minor Cycles per iteration	171	37	37	17

```
                                  02 15 66                 PAGE NUMBER     2
FLUSH  SA1 I
       SA2 K
       PX3 B0 X2                          .BEGIN FLUSH INNER DO
       SA4 UN10
       DX5 X4*X3
       UX0 B0 X5
       IX2 X1+X0
       SA3 A-11+X2
       SA1 K
       SA2 J
       PX4 B0 X2
       SA5 UN10
       DX0 X5*X4
       UX2 B0 X0
       IX4 X1+X2
       SA5 B-11+X4
       FX0 X3*X4
       SA1 I

       SA2 J
       PX3 B0 X2

       SA4 UN10
       DX5 X3*X4
       UX2 B0 X5
       IX3 X1+X2
       SA4 C-11+X3
       FX5 X0+X4
       NX6 B0 X5
       SA1 I

       SA2 J
       PX3 B0 X2

       SA4 UN10
       DX5 X3*X4
       UX0 B0 X5
       IX2 X0+X1
       SA6 C-11+X2

       SA1 K
       SX6 X1+1
       SA6 K
       SX0 10
       IX1 X0-X6
       PL  X1 FLUSH                       .END OF FLUSH INNER DO
```

```
         PS
         TIME IN
         PL X1 STD
STD      SA1 A-11+B1                      . BEGIN STD INNER DO
         SA2 B-11+B2
         SA3 C-11+B3
         SB1 B1+10
         SB2 B2+1
         FX0 X1*X2
```

```
                    02 15 66            PAGE NUMBER      3

         SB4 B4+1
         FX5 X0+X3
         NX6 B0 X5
         SA6 C-11+B3
         NE  B0 B4 STD                    .END OF STD LOOP
         PS

         PL X1 ABL

STDA     SA1 B1                           . BEGIN STDA INNER DO
         SA2 B2
         SA3 B3
         FX0 X1*X2
         SB1 B1+B4
         SB2 B2+B5
         SB6 B6+B5
         FX5 X0+X3
         NX6 B0 X5
         SA6 B3
         NE B6 B0 STDA                    .END OF STDA LOOP
         PS

         PL X1 ABL

FLASH    FX0 X1*X2                        . BEGIN FLASH INNER DO
         SB3 B3+B4
         SA1 A1+B1
         SA2 A2+B2
         FX3 X6+X0
         NX6 B0 X3
         NE B3 B0 FLASH                   .END OF FLASH LOOP
         END
```

APPENDIX "B"

I. Cost Objectives

A. Development, Documentation, and QA costs.

Fiscal Year by Quarters	Development	Documentation	QA	Total
1 Q 67	$ 41,000	$ 3,600	$	$44,600
2 Q 67	62,000	6,000		68,000
3 Q 67	67,000	9,100		76,100
4 Q 67	68,000	9,300	8,000	85,300
1 Q 68	37,000	---	9,000	46,000
TOTALS	$275,000	$28,000	$17,000	$320,000

II.

Schedule Objectives	SO/CD	Present Estimate	6000L Product Plan Date*
SIR approved		6/14/66	
DO submitted	CD	6/20/66	
GED submitted	CD	7/05/66	
ERS submitted	SO	9/05/66	
IRS completed	SO	11/01/66	
Product Demonstration	SO	4/11/67	
QA Test Cases to Project	SO	4/10/67	
Product Submitted to QA	SO	6/02/67	
Product Released to field	SO	8/11/67	
GIM published	SO	10/16/66	
Reference Manual published	SO	4/17/67	

* The March Plan proposed a SCHEDULE OBJECTIVE for Product Release in 2nd. Q/67. However, in the March Plan this product was relegated to further study with attendant potential extension of release date.

COMPANY PRIVATE

J) MIMMACK

CODING GUIDELINES

1. Critical loops which require different coding for the 6400 than the 6600 should be coded both ways and preceded by IFF and IFT pseudo ops. The assembly control parameter must appear prior to any executable code in the subprogram and will have the name SIXTY e.g. SIXTY EQU 4 indicates assembly for the 6400.

2. There will be a cell named SIX000 in the main control program indicating which machine code is being generated for named SIX000.

3. Blocks of remarks heading subroutines or other sections of code should be used to specify such things as the function of the section of code and calling sequence required.

4. Comments accompanying every few instructions should describe the operations being performed.

5. Remarks and comments must be updated as the coding is changed so that it is possible at all times to read a listing mainly by commentary rather than by instructions.

6. To minimize difficulties of debugging and future modifications of the system, straight forward coding is generally preferable to subtle or involved coding. Whenever such coding is required justification must be included in the task documentation along with adequate remarks to warn and guide the unsuspecting reader through the proper analysis.

7. Symbols used in source code should be identical to those used in flow descriptions and should have some mnenmonic content, if possible.

8. As much of the compiler as is reasonable should be written in FORTRAN. This may require the writing of several machine language functions resulting in a mixed approach. This is generally preferable to an all assembly language product. The common subset of FORTRAN Version 1.0 and 3.0 must be used.

9. All linkage connecting an assembly subprogram to FORTRAN subprogram should be done through macros so that the transition between compiling the

compiler under different compilers will be simplified.

10. Where a label is required and no meaningful label is used, use the first and last characters of the subprogram name, two of your initials and two digits.

TASK DOCUMENTATION

Task Description

The description consists of a list of the functions performed by the task, the interaction of the task with any others, and any possible side effects. As development progresses, description of tasks (and any subroutine within the tasks) are expanded to include specification of calling sequences, changes to the contents of core and registers, I/O equipment usage, space requirements for code and data, table formats, and possible diagnostics, and error actions.

Method

Methods used to accomplish the task are added to the documentation as soon as they are known and are updated when any change is to be made. Any special algorithms for such things as conversions or table manipulation should be included.

Memory Layout

The memory layout for a task, or a subroutine within a task, consists of a list of all symbols used for referencing data and a descirption of the significance of each item in the coded program. Whenever possible, attempt should be made to use symbols with some mnemonic value. These same data referencing symbols must also be used in the flow description and coded program.

Flow Description

Written descriptions of the overall flow of the task and of each subroutine in the task must be approved by the project manager before coding begins and must be updated as checkout proceeds. The flow description may have the form either of a flow chart or a narrative flow description. In either case, the symbols used should be identical to those appearing in the coded program. If flow charts are used, flow charting standards given in the Programming Handbook should be followed. If narrative flow descriptions are used, attempt should be made to use standard printer characters and to relate the wording of the description to the wording of the remarks and comments in the program listing.

64/6600 FORTRAN VERSION 3.0

LIST OF TASKS

COMPILER

1. COMMON

RW	CONTROL	Main controlling routine
RW	ERRORS	Error diagnostics reorder
JM	SCANNER	Reads and lists Source statements, and forms ELIST.
RW	SIO	Output R-list to intermediate storage and handles all I/O
RW	FORMAT	Processes FORMAT statements
CN	LISTPROC SYMBOL	Search-and-entry routine for SYMTAB and memory management

2. PASS 1, PHASE 1

JM	PHS1CTL	Controlling routine for phase 1
JM	DECPRO	Processor for declarative statements

3. PASS 1, PHASE 2

RW	PHS2CTL	Controlling routine for phase 2. It includes BCD to binary conversion routines.
TM	ASF	Arithmetic statement function processor
JM	DATA	DATA statement processor
HR ~~PT~~	DOPROC	DO-loop processor for pass 1
TM	ARITH	Processor for arithmetic, logical and masking statements.
TM	IF	Processes all IF statements
RW	LISTEDIO	Processes I/O statements
CN	CALL	CALL statement processor
CN	ASSIGN	ASSIGN statement processor
CN	GOTO	Processes all GOTO statements
RW	TAPES	Processes tape handling statements: ENDFILE, BACKSPACE, REWIND
TM	INDXFN	Generates macro references for index functions
CN	ENTRY	ENTRY statement processor
CN	RETURN	RETURN statement processor
CN	END	END statement processor

4. PASS 2

RW	ERROR 2	Issues error messages
FT	PRE	Pre-optimizer, pre-processes R-list for OPT
FT	OPT	Code optimizer and generator
FT	POST	Post optimizer, generates assembly code

OBJECT TIME

CN 1. Mathematical Library Routines:

RW 2. I/O Routines

OTHER

FT 1. R-list - Definition of intermediate language