A TRANSLATOR WRITING SYSTEM FOR MICROCOMPUTER HIGH-LEVEL LANGUAGES AND ASSEMBLERS

W. Robert Collins*
Computer Sciences Corporation
Hampton, Virginia

John C. Knight
Langley Research Center
Hampton, Virginia

and

Robert E. Noonan**
College of William and Mary
Williamsburg, Virginia

NASA LaRC uses many dedicated microprocessors in aerospace research. Few software tools are available for these machines, and in particular, very few have any form of high-level language facility. Since the Langley environment involves considerable experimentation, a great deal of software is experimental and may change frequently. It has to be prepared relatively quickly and at low cost.

In order to implement high-level languages whenever possible, a Translator Writing System of advanced design has been developed. It is intended for routine production use by many programmers working on different projects. As well as a fairly conventional parser generator, it includes a system for the rapid generation of table driven code generators. This code generation system is the result of research performed at the College of William and Mary under NASA sponsorship. The parser generator was developed from a prototype version written at the College of William and Mary.

The Translator Writing System includes various tools for the management of the source text of a compiler under construction. In addition, it supplies various "default" source code sections so that its output is always compilable and executable. The system thereby encourages iterative enhancement as a development methodology by ensuring an executable program from the earliest stages of a compiler development project.

This presentation will describe the Translator Writing System and some of its applications. These include the PASCAL/48 compiler, three assemblers, and two compilers for a subset of HAL/S. PASCAL/48 is a Pascal-like language for the Intel-8748 microcomputer. The assemblers which have been built are for assembly language subsets for the Intel-8080, the Motorola M68000, and the NSSC-II. The HAL/S subset was implemented for the Intel-8080 and the GE 703. Detailed measurements of the use of the system to build the code generators for the HAL/S compilers will be given.

*Work performed under NASA contract numbers NAS1-14900 and NAS1-16078.

**Work performed under NASA grant number NSG-1435.
THE PROBLEM

- NEED HIGH-LEVEL LANGUAGES, HENCE COMPILERS
- NEED ASSEMBLERS
- ONE SOLUTION IS A TWS

TWS CRITERIA

- ENCOURAGE ITERATIVE ENHANCEMENT
  - EARLIEST POSSIBLE EXECUTION
  - TEXT MANAGEMENT TO RELIEVE TEDIOUS
- FLEXIBILITY IN ITS USE
- TRANSPORTABLE IMPLEMENTATION

TRANSLATOR WRITING SYSTEM
USE OF TWS

1. IF PARSER NEEDED, RUN PARGEN, EXECUTE RESULTING COMPILER TO TEST.

2. CHANGE GRAMMAR AS NECESSARY, RERUN PARGEN.

3. ADD SEMANTICS USING EDITOR.

4. RECOVER GRAMMAR AND SEMANTICS WITH GRAMGEN IF NECESSARY TO RERUN PARGEN.

5. IF CODE GENERATION NEEDED, PREPARE CGGL SPECIFICATION AND RUN CODGEN.

6. MODIFY CGGL AS NEEDED.

7. ITERATE THROUGH ABOVE STEPS ADDING LANGUAGE FEATURES AS DESIRED.

PARGEN

• INPUTS
  - GRAMMAR IN STANDARD BNF
  - SEMANTICS IN PASCAL
  - SKELETON OR OLD COMPILER

• OUTPUT IS AN EXECUTABLE COMPILER INCLUDING
  - SCANNER
  - LALR (1) PARSER
  - SEMANTICS ROUTINE

• TEXT MANAGER PRESERVES PROGRAMMER'S CONTRIBUTION TO COMPILER E.G., SYMBOL TABLE ROUTINES.
CODGEN

• INPUTS
  - CGGL SPECIFICATION
  - SKELETON OR OLD COMPILER

• OUTPUT IS AN EXECUTABLE COMPILER INCLUDING A CODE GENERATOR.

• CGGL IS A NON-PROCEDURAL LANGUAGE FOR DESCRIBING THE CODE-
  GENERATION PROCESS.

• TEXT MANAGER PRESERVES PROGRAMMER'S CONTRIBUTION TO COMPILER
  E.G., MACHINE LANGUAGE FORMATTER.

PASCAL/48

• INTEL-8748
  - MICROCOMPUTER
  - 8-BIT CPU
  - 64 WORD RAM
  - 1024 WORD ROM
  - 27 I/O LINES

• PASCAL/48
  - PASCAL DERIVATIVE FOR 8748
  - EXTENSIONS TO ALLOW CONTROL OVER GENERATED CODE
  - RESTRICTIONS TO PROHIBIT INEFFICIENT FEATURES
  - COMPILER AVAILABLE ON CDC CYBERS
ASSEMBLERS

- CUSTOMIZED SKELETON FOR ASSEMBLERS
  - TWO PASSES
  - STANDARD LISTING BY DEFAULT
  - FLEXIBLE INPUT FORMAT CONVENTIONS
  - HANDLES MACROS WITHOUT PARAMETERS

- COMPARED TO META-ASSEMBLER, ASSEMBLER BUILT FOR NSSC-II
  - WAS PRODUCED MORE QUICKLY
  - EXECUTES 5 TIMES FASTER
  - USES ONE FOURTH THE SPACE

EXAMPLE PASCAL/48 PROGRAM

```
1 PROGRAM FOR_YOU;
2   VAR I[2] : INTEGER;
3       A[16..300, PDM] : ARRAY [100] OF INTEGER;
4
5 VALUE A = (99 OF 0, 1);
6
7 PROCEDURE GET_INPUT;
8 BEGIN
9   REPEAT
10      UNTIL PORT1 BIT 3
11   END; (* GET_INPUT *)
12
13 BEGIN (* PROGRAM FOR_YOU *)
14   FOR I := 100 DOWNTO 1 DO
15   BEGIN
16      GET_INPUT;
17      PORT1 := PORT1 AND 2..1100011;
18      PORT2 := A[I] + PORT1 XOR I
19   END (* FOR I := 100 DOWNTO 1 DO BEGIN *)
20 END. (* PROGRAM FOR_YOU *)
```
GENERATED CODE FOR EXAMPLE PROGRAM

```assembly
L003: JMP L009
NOP
L007: JMP L007
CLR A
MOV PSW,A
JMP L012
L009: JMP L009
L00D: IN A,P1 ; LINE 9
CPL A ; LINE 13
JZ3 L000 ; LINE 13
RET ; LINE 13
L012: MOV R2,#99 ; LINE 14
L014: CALL L000 ; LINE 18
ANL P1,#227 ; LINE 21
IN A,P1 ; LINE 22
MOV P1,A ; LINE 22
MOV A,P2 ; LINE 22
MOVP3 A,#2A ; LINE 22
ADD A,P2 ; LINE 22
XRZ A,R1 ; LINE 22
OUTL P2,A ; LINE 22
DJNZ R2,LC14 ; LINE 22
```

SEPARATE CODE GENERATION
USING CGGL

LANGUAGE: HAL/S

INTERMEDIATE CODE LANGUAGE: HALMAT

--- 178 OPERATORS TOTAL
--- 30 OPERATORS IMPLEMENTED
--- 25 GENERATE CODE
--- BASICALLY AN INTEGER SUBSET WITH SIMPLE CONTROL STRUCTURES

CODE GENERATORS
--- ONE PASS
--- NO PRE-OPTIMIZATION PASS
--- NO PEEPHOLE OPTIMIZATION
--- INTEL 8080, GE 703
IMPLEMENTATIONS

INTEL 8080

— 8 BIT MACHINE
— SINGLE ACCUMULATOR
— NO INDEX REGISTER
— 1, 2, 3 BYTE INSTRUCTIONS
— HARDWARE STACK
— ONLY INTEGER ADD, SUBTRACT
— 16 BIT ADDRESSES

GE 703

— 16 BIT MACHINE
— SINGLE ACCUMULATION
— INDEX REGISTER
— ONE WORD INSTRUCTIONS
— NO HARDWARE STACK
— INTEGER ADD, SUBTRACT, MULTIPLY, DIVIDE
— ONLY ADDRESS CURRENT PAGE, PAGE ZERO
— PAGE: 256 WORDS

703 Code Generator

<table>
<thead>
<tr>
<th>TASK</th>
<th>TIME (DAYS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>READING MANUAL</td>
<td>.5</td>
</tr>
<tr>
<td>CGGL PROGRAM</td>
<td>1.5</td>
</tr>
<tr>
<td>WRITING PASCAL ROUTINES</td>
<td>1.5</td>
</tr>
<tr>
<td>DEBUGGING</td>
<td>1.0</td>
</tr>
</tbody>
</table>

4.5 DAYS

Notes:

1. ALL PROGRAMS WERE CODED AND KEYED BY NOONAN.
2. SOME OF DEBUGGING TIME WAS USED IN CLEANUP.
3. ONE DEBUGGING RUN WAS USED TO FIX A BUG INTRODUCED BY CLEANUP.
4. A TOTAL OF 6 RUNS (EXECUTION) WERE USED.
5. ONE CGGL BUG.
### 703 Implementation

<table>
<thead>
<tr>
<th>Source of Code</th>
<th>No. Procedures</th>
<th>% Lines</th>
<th>% Instr. Storage</th>
</tr>
</thead>
<tbody>
<tr>
<td>8080 Imp.</td>
<td>46</td>
<td>58%</td>
<td>58%</td>
</tr>
<tr>
<td>Modified 8080</td>
<td>4</td>
<td>8%</td>
<td>6%</td>
</tr>
<tr>
<td>Noonan</td>
<td>9</td>
<td>10%</td>
<td>10%</td>
</tr>
<tr>
<td>CGGL</td>
<td>1</td>
<td>24%</td>
<td>26%</td>
</tr>
</tbody>
</table>

**Notes:**

1. **CGGL program:** 292 lines
2. **Pascal program:** 890 lines
3. For an earlier non-table-driven implementation, CGGL accounted for 83% of lines and 77% of storage.