Examples of LITTLE-generated Code

In an earlier newsletter (LITTLE news No. 20) we gave some examples of the LITTLE source code from the SETL Run-Time Library (SRTL); in this newsletter, we give some examples of the 6600 machine code produced by the current LITTLE compiler for sample parts of the SRTL. For each example, we present the LITTLE source (after macro expansion) and a COMPASS-like representation of the machine-code generated by LITTLE. Note that the use of a '+' character in the label field indicates that the instruction begins a new word.

Ex. 1--Simple Stores
The LITTLE source

T = 5000 - MAXZZYZ; TRES = T; RUNNINGBLK = T;

compiler into

\[
\begin{align*}
&\text{SX1} \quad 5000 \\
&\text{SA2} \quad \text{MAXZZYZ} \\
+ &\text{IX3} \quad X1 - X2 \\
&\text{BX6} \quad X3 \\
&\text{SA6} \quad T \\
+ &\text{SA6} \quad \text{TRES} \\
&\text{SA4} \quad T \\
+ &\text{BX7} \quad X4 \\
&\text{SA7} \quad \text{RUNNINGBLK}
\end{align*}
\]

This code takes about 60 minor cycles (a minor cycle is 100 nanoseconds) to execute, and requires four words. The preferred code, which takes about 35 minor cycles, and three words, is

\[
\begin{align*}
+ &\text{SA2} \quad \text{MAXZZYZ} \\
&\text{SX1} \quad 5000 \\
+ &\text{IX6} \quad X1 - X2 \\
&\text{BX7} \quad X6 \\
&\text{SA6} \quad T \\
+ &\text{SA7} \quad \text{TRES} \\
&\text{SA6} \quad \text{RUNNINGBLK}
\end{align*}
\]
Ex 2-- Argument access and branching

The LITTLE source

```
SUBR GETSTG (N,P);
IF ( N.EQ.0 ) GO TO A09;
GO TO B09;
/A09/ CALL ABORT; /B09/ ...
```

compilers into

```
CALLAD BSS 1 .arg-list address planted
               .here as part of call
GETSTG BSS 1 .entry/exit word
+ SA1 CALLAD
  SA2 X1 + 0
+ SA2 X2 .X2 = value of N
  MX3 0 .X3 = 0
  BX4 X2-X3 .equality test
  SB2 B0
+ NZ X4,  L01 .full-word no-ops
  SB2 1
L01 NE B2, B0, L02
+ JP B09
L02 NO
+ SA0 ABORT-1
  RJ ABORT
B09 ...
```

This code takes seven words, and executes in about 80 cycles
(if ABORT is called). The preferred code, requiring three words
and about 35 cycles (if ABORT called) is

```
GETSTG BSS 1
SA1 CALLAD
SA2 X1 + B0
SA2 X2 + B0
+ NZ X2, B09
RJ ABORT
```
Ex. 3--multiplication by 1

Though the LITTLE source is not available, the following code-fragment (from routine START) was observed:

```
SA1  T
+  SX0  1
PX0  X0, BO .Pack
PX1  X1, BO
+  DX1  X1*X0 . multiply
UX1  X1, BO .get integer product
SA2  X1 +U .ie, U(x1)
+  MX3  0
BX6  X3
SA6  A2
+  SA1  T
SA2  1
+  IX3  X1+X2
BX6  X3
SA6  A1
```

The source is probably

\[ U(T*1) = 0; T = T + 1; \]

The preferred code is

```
+  SA1  T
MX6  0
+  SA6  X1 + U
SX7  X1 + 1
+  SA7  T
```

The longer code takes about 60 minor cycles, the shorter about 30.
Ex. 4-- simple loop

The LITTLE source

\[
\text{TEMP} = 1; \\
\text{/A0l/ IF (TEMP .GT. 5000)GO TO B0l;} \\
\quad \text{STORAGE (TEMP)} = 0; \\
\quad \text{TEMP} = \text{TEMP} + 1; \\
\quad \text{GO TO A0l;} \\
\]

which is a "memory-set" loop, compiles into

\[
\begin{align*}
&\text{SX1} \quad 1 \\
&\text{BX6} \quad \text{X1} \\
&+ \quad \text{SA6} \quad \text{TEMP} \\
&A0l \quad \text{SA1} \quad \text{TEMP} \\
&\quad \text{SX2} \quad 5000 \\
&+ \quad \text{IX3} \quad \text{X2} - \text{X1} \\
&\quad \text{SB1} \quad \text{BO} \\
&+ \quad \text{PL} \quad \text{X3, LOl} \\
&\quad \text{SB1} \quad 1 \\
&\text{LOl} \quad \text{NE} \quad \text{B1,BO,B0l} \\
&\quad \text{SA1} \quad \text{TEMP} \\
&+ \quad \text{SB2} \quad \text{X1} \\
&\quad \text{SA2} \quad \text{B2 + STORAGE} \\
&\quad \text{MX3} \quad 0 \\
&+ \quad \text{BX6} \quad \text{X3} \\
&\quad \text{SA6} \quad \text{A2} \\
&+ \quad \text{SA1} \quad \text{TEMP} \\
&\quad \text{SX2} \quad 1 \\
&+ \quad \text{IX3} \quad \text{X1 + X2} \\
&\quad \text{BX6} \quad \text{X3} \\
&\quad \text{SA6} \quad \text{A1} \\
&+ \quad \text{JP} \quad \text{A0l}
\end{align*}
\]

Since loop takes more than 7 words, it doesn't fit in the stack. Since TEMP not used outside loop, it need not be stored, also TEMP used as a subscript and thus may be kept in B-register.
Thus preferred code is

\[
\begin{align*}
SB1 & \quad 1 \\
SB2 & \quad 5000 \\
SB3 & \quad 1 \\
MX6 & \quad 0 \\
SA6 & \quad \text{STORAGE + B1} \\
L & \quad \text{SB1 + B3} \\
SA6 & \quad A6 + B1 \\
LT & \quad B1, B2, L \\
\end{align*}
\]

The shorter code fits in stack (main loop is a single word), and requires about three words for entire loop instead of nine for longer LITTLE generated code.

Note the perhaps the best way to handle storage-set loops is to call a storage-set function which is carefully hand coded to fit in stack and stores both \text{STORAGE (TEMP)} and \text{STORAGE(TEMP+1)} in single pass thru loop (such a routine is available) similar remarks apply to storage-move loops.

In summary, the example show that a relatively simple improvement in code-generation would probably reduce both code-size and execution time by a factor of two for typical LITTLE programs. Furthermore, most of these improvements could be done in a separate job step, which takes as input the code produced by LITTLE and produces improved code (on a subroutine by subroutine basis). This separate code-improver could be developed without any substantial change in the current LITTLE compiler; the only change required in the LITTLE compiler is the ability to produce symbolic, COMPASS-like output, instead of loader input modules (otherwise, the code-analyser must unpack the loader tables).