MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Reply to: Project MAC
545 Technology Square
Cambridge, Mass. 02139

Telephone: (617) 864-6900 x6201

January 29, 1968

TO: F. J. Corbató and C. T. Clingen

FROM: J. H. Saltzer

SUBJECT: Effect of EPL on Traffic Control module.

R. L. Rappaport has reported that execution time for a complete trip through the modules of the Traffic Controller requires about 10 ms. Conversations have not revealed any obvious insight as to why this time should be so large. This weekend I made a brief study into one traffic controller module to see what could be learned. The results indicate some of the problems of writing efficient code with the EPL compiler.

The module chosen for study was the external procedure named block. This procedure constitutes 38 EPL statements including calls to other internal procedures of the Traffic Controller but not including the statements of those internal procedures. These 38 statements compile into 339 locations full of EPLBSA instructions, and thus constitute about 20 per cent of the 1655 instructions of the segment px. (The only parts of the Traffic Controller not included in px are the scheduler module and the EPLBSA stack switching module used to enter px.)

Since for the most part the Traffic Controller has neither loops nor long sequences rarely executed, we may take as an approximation that its execution time is proportional to its length. This is especially true in the block procedure, which has no loops and only a couple of minor branches. We therefore examine carefully the reasons why the length of the block module is what it is.

The first observation is that of the 38 statements of block, 17 are concerned with setting on and off individual process state locks. Using instead a single global interlock for the Traffic Controller, these statements can be condensed to four; over 1/3 of the EPLBSA instructions disappear at the same time. Assuming that this redesign were made, the following statistics (obtained by counting instructions rather than by recompiling) hold:

Original length of block	339
Extra locking statements	- 103
Length of block with global interlock	236

A second observation is that the EPLBSA code is filled with an unseemly number of and instructions and specifier manipulation, which trace directly to bit string arguments and data items. Inspection showed that all of the bit strings concerned were either 1-bit items stored in a whole word, or 72-bit strings stored in a word pair, so one could declare them fixed, with no loss in storage space. Doing this results in the following counts:

Length of block with global interlock	236
Instructions due to strings	- 57
Length of block with no strings	179

A third observation is that a surprising length of code was required to compute a pointer into the active process table from an index into the same table, viewed as an array. This was done twice here, as well as elsewhere in the process exchange, and can be avoided by storing the pointer in the process data segment after computing it once in swap DBR. This redesign yields the following result:

Length of block with no strings	279
Extra instructions for pointer computation	- 33
Length of respecified block	146

At this point, two bench mark experiments were made. The first was to write a transliteration of the respecified block in MAD for the 7094, using COMMON in place of EXTERNAL, and a number of similar transformations. The second experiment was to write a version of the respecified block in EPLBSA using all tricks available to make the program as short as possible. These two coding efforts compare with the EPL program as follows:

Respecified block in EPL	146
Respecified block in MAD (7094)	136
Respecified block in hand coded EPLBSA	53

One may conclude from these figures that the EPL compiler is not doing such a bad job on this particular program.

Finally, an instruction-by-instruction comparison between the EPLBSA program produced by EPL and the EPLBSA program produced by hand was made, to identify what it was that the compiler found difficult or awkward to compile efficiently. The following table summarizes the apparent sources of the 93 "extra" instructions inserted by EPL.

Unneeded garbage, never executed	5
Unneeded set up on calls with no arguments	4
Reloading of an already loaded register	14
Didn't save ap, had to reload	2
Sloppy use of base registers	4
Lack of access to RMSK, SMSK, STAC instructions	34
Didn't use clever 645 instruction in special case	8
Didn't optimally reorder comparison	1
Extra transfers	5
External calls to internal subroutines	13
Used a subroutine for the save sequence	3
	93

Interpretation of this last set of data is difficult, since it represents a rather small sample. One point is suggested by the data, however: the Traffic Controller may be significantly slowed down by need to perform many external subroutine calls to get at special hardware instructions. (In fact, the hand-coded EPLBSA program does more than the EPL-coded program, since it actually executes the instruction in question, while the EPL-coded version merely calls the procedure, which must go through further fol-de-rol to untangle the calling sequence before executing the special instruction.)

As a purely practical matter, this observation suggests a further respecification of block: the setting of the global interlock and associated processor interrupt mask should be done in the EPLBSA stack switching interlude procedure which calls block. (Argument copying to wired-down core must also be done there.) This redesign removes 16 needed instructions from both our EPL-produced and our hand produced program, as well as the 34 extras in the EPL version, leaving us with

EPL procedure without interlock, mask setting, or argument copying	146 - 50 = 96
EPLBSA hand-coded to same specs	53 - 16 = 37

Overall, there is the impression that the remainder of the program has been "nickled and dimed" to death, and that there are no spectacular simple performance improvers to suggest for the compiler. It does appear that performance of the compiler is now sufficiently good that simple register optimization has become a significant issue. Between reloading already loaded registers, not saving the argument pointer, and not using base registers most effectively, 20 instructions were added to what could be a 37 instruction program. This statistic suggests that ultimately register optimization will be quite important. On the other hand in the present situation, with other extras being inserted from a variety of sources, the effect is that 20 out of 96 instructions were the result of lack of optimization of register usage. Thus the immediate effect of the lack of register optimization is only moderate, especially since squeezing all 20 of the extra instructions out may be difficult or impossible.

If one were so confident as to hazard the prediction that similar results hold for the entire Traffic Controller, one would expect to find that redesigns of the original EPL program to optimize use of the present compiler could drop the size, and therefore the running time, to about 40 per cent of their present values, as it did for block. Further reduction of about a factor three is potentially possible by moving completely to machine language. These two effects combine to bring the present 10 ms. round trip time down into the area of 1.5 ms. After making allowance for the possibility that the block module is somehow more susceptible to optimization than the rest, one might still assume that the figure of 2.0 ms. is an upper bound on the intrinsic computation requirements of the traffic control modules.