From: W. Olin Sibert

To: MTB distribution

Date: February 1, 1981

Subject: Constructing Programs for Multiple System Types

INTRODUCTION:

The development of Multics software for the ORION requires that facilities be available in the language translators and runtime environment that permit programs to be compiled differently or run differently depending on the type of system they are being used with. Three forms of selection will be used: selection at runtime, by executed code, selection at compile time, by macro expansion and compiler option, and selection at the time the system is built, from entirely separate programs. The actual user interfaces to these are described in the Implementation section, later, and some of the reasons for decisions about the implementation are described in the Other Issues section at the end.

Of these, all mechanisms except the selection of compile time optimizations and code generation choices will be used primarily by programs in the supervisor. User programs will not generally need to make choices based on system type at runtime, and any user source program will run on any type system as long as it has been compiled for that type of system. This means that all selection on system type will be transparent to writers of user code, with few exceptions; a user program will run on the system it was compiled on, without changes or any explicit specification of system type.

On the other hand, in the supervisor, the choices become more explicit. Wherever possible, when it will not pose a meaningful cost in execution speed, all choices will be made on the basis of runtime checks. When this is inappropriate, macro expansion will be used, as will the creation of entirely separate programs. The choices made in the supervisor must also be more fine grained than the choices made for user programs. The supervisor may sometimes be concerned with the difference between two different types of SCU, whereas the code generated for user programs need only be selected on the basis of broad classifications of system architecture.

In fact, the only selections which need to be performed for most user code are those which deal with instruction optimization and availability. We have two major classes of these today: the Level 68 and the ORION. Although the DPS/8 is an entirely new piece of hardware, the differences it presents to user programs are completely insignificant. We can expect this trend to continue indefinitely; there may be many different models to choose from, of widely varying performance, but they will fall into just a few broad classes of instruction sets. For this reason, the transparent selection mechanisms we provide for user program writers can be very simple; their choices are few, and the goal of having programs run on the system on which they were compiled is readily achievable.

In order to accomodate migration, of course, it will be necessary to provise users a mechanism to say "compile this as if it were being compiled on the FritzBlatt model 53", but even this will be little used. Almost all user programs will be used on the same systems where they were compiled, and their writers need not be concerned with any of these issues.

When writing supervisor code, the same choices must be made for instruction set usage, but other, more explicit choices must be made as well. Since we can assume that programmers writing supervisor code are familiar with the need for this, all other choices must be made explicitly. It is likely, for instance, that many supervisor programs will not be compiled on the same system where they are to run.

IMPLEMENTATION OF SYSTEM SPECIFICATIONS:

In order to provide these selection functions, the following facilities will be available:

1) There will be an include file describing the canonical integer values for system type. Initially, there will be two such types: L68_SYSTEM, and ADP_SYSTEM, as well as a value specific to no system at all, ANY_SYSTEM. Others may be added in the future as needed. It will be called system_types.incl.pl1. The value for the type of the current system will be available from the external variable sys_info$system_type.

2) There will be a subroutine which translates string specifications of target machine, appearing as control argument or conditional compilation operands, into their canonical integer form. It will also supply the type of the running system for use as a default. This is described in MTB-495.

3) The language compilers (pl1, fortran, cobol) will all be able to generate code which will work on all types of system. Initially, this can be done by generating code which will work on all systems, but optimizations for specific systems can be added later.

   By default, all translators will produce code appropriate to the system on which they are running. A "-target_machine" control argument will be available to specify code generation for other system types. In order to get code which can be executed on any system, a target machine of "any" must be specified.

4) The ALM assembler will be able to assemble all the instructions available on all the systems. This is provided by the multiple decor support, detailed in MTB-469. By default, ALM will assemble instructions in a decor which can be executed on all types of systems, but this can be overridden either by use of the the "decor" pseudo-op, or by explicit specification with the "-target_machine" control argument. The appearance of a "decor" pseudo-op in the source will override any default or control argument specification.

5) The ALM assembler will provide a conditional assembly facility based on the argument to the "-target_machine" pseudo-op, implemented as "iftarget STR" and "ifntarget STR", where STR is a string acceptable to the canonicalization subroutine described above. If these constructs are used, the "-target_machine" control argument must also be specified; there is no default, since this construct will be used primarily in supervisor programming, and should not have accident-prone defaults. Note that these pseudo-ops can also be used to cause temporary changes of decor while assembling system-specific portions of code.

6) The PL/I macro expander will provide a system type predicate function for use in conditional expansion, of the form "%target (STR)". As with the ALM feature, if "%target" is used, the "-target_machine" control argument must also be specified. It will be a severity two error to use "%target" if none was specified, but it will assume a default of the current system in order to allow the expansion and compilation to continue.

7) These features of the PL/I macro expander will be used to make include files describe structures appropriate to particular machines. For instance, there will be three include files describing the format of an SDW: sdw.l68.incl.pl1, for the Level 68, sdw.adp.incl.pl1, for the ORION, and sdw.incl.pl1, which will contain conditional compilation directives to expand to either of "%include sdw.l68;" or "%include sdw.adp;", whichever is appropriate for the target system specified on the command line.

Because the initial implementation of the macro processor will be standalone, this will require that include files containing that sort of conditional expansion be included with a "%INCLUDE" statement, rather than with the usual "%include". This deficiency will disappear when the macro processor is integrated into the compiler, although it will still be necessary to specify a target system on the command line in order to use such include files.

The combination of defaults and required specifications in these mechanisms provides a convenient and error-free way for users to compile their programs without worrying about the system type distinctions. It also will make supervisor programming less accident-prone, by requiring that programs with specific requirements have those requirements answered by explicit specification, rather than by probably inappropriate default.

OTHER ISSUES:

There will be no specific system type for the DPS/8, because it is so similar to the Level 68. In fact, in the anticipated mixed and L68 compatible configurations, the differences are virtually unnoticeable. In native-mode configurations, utilizing the hardware cache coherency mechanisms, the differences will be greater, but still confined to small portions of the supervisor where the choices will be made at runtime.

There is no provision for specifying multiple models of a system in this scheme. While this is not a problem now, there may come a time when there are different models of basically the same architecture, but which could benefit from compiler optimizations specific to that particular model. These optimization criteria will not affect existing code, or require any recompilation, but if it is desired to take advantage of these possible optimizations, the target specification will be extended to accept strings of the form "ORION.4X", "ORION.10X", etc. This will be completely compatible with existing usage. It seems likely that this sort of specific optimization will never be required, since the amount of effort required to implement it could better be expended implementing effective global optimizations which would have a higher payoff; still, one never knows.

The default for the language translators is different from that implemented by ALM for two reasons. One is that programs written in ALM are likely to require very specific changes based on system type, since they otherwise would be written in pl1, and the other is one of convenience for user programs. The entire supervisor will either have to be compiled with a target of "any" if it is to run on any machine (except where specific requirements exist, of course), or with a specific target to generate code which is optimized for that machine.