Release Notes for HI-TECH C Compiler V4.11

October 1989

1. Introduction

This release of HI-TECH C includes some new features and a few bug fixes.

There are several new error and warning messages, and some existing messages have been improved. Error handling generally is better. The major new features are interrupt functions and port data types.

2. Interrupt Functions

The compiler incorporates features allowing full interrupt handling without any assembler code. The type qualifier *interrupt* may be applied to a function to allow it to be called directly from a hardware (or software) interrupt. This involves performing a return from interrupt rather than a return from subroutine at the end of the function. Any necessary registers are saved and restored as required.

An interrupt function must be declared void and may not have parameters. It may not be called from C code, but it may call any function itself. An example of an interrupt to service the RSTC interrupt on the NSC800 follows.

```
long count;

void interrupt
real_time(void)
{
        count++;
}
```

To set interrupt vectors the routine *set_vector()* is provided. The arguments to *set_vector* are the address of the interrupt vector, casted to a pointer to a pointer to an interrupt routine, and the address of the interrupt function.

This routine is declared in the header file **<intrpt.h>**, along with macros to enable and disable interrupts. These are *ei()* and *di()* respectively. An example follows of setting a vector for the

RSTC interrupt on the NSC800.

```
#include  <intrpt.h>
#define RSTC    ((isr *)0x002C)

static void interrupt
service_int(void)
{
        tcnt++;
        /* clear interrupt source here */
}

static void
setup_interrupt(void)
{
        di();
        set_vector(RSTC, service_int);
        /* setup interrupt source here */
        ei();
}
```

See the description of the function *set_vector()* below also.

3. Port Data Types

Those processors that have a separate I/O space now allow ports to be accessed directly via port data types. It is now possible declare variables of port type, as well as pointers (including constant pointers) to ports. For example:

```
        port unsigned char *    pptr;
        port char               io_port @ 0xE0;
```

The variable "pptr" is a pointer to an unsigned char in the port data space. The variable "io_port" is a character mapped directly onto port 0E0H and will be accessed using the appropriate IN and OUT instructions. If the address of a port is an absolute value, you can use a typecast absolute address, for example:

```
#define IO_PORT     (*(port unsigned char *)0xE0)
func()
{
     IO_PORT = 0x40;
}
```

defines a port at the absolute address of 0xE0. Note that in this case, it would have been preferable to use:

```
port unsigned char  IO_PORT @ 0xE0;
```

IO_PORT can then be used just like a variable as shown above. The code shown would generate the following code:

```
     LD   A,40H
     OUT  (0E0H),A
```

4. Absolute Variables

This release of the compiler also includes a facility which allows variables to be placed at absolute addresses. This is done using the @ initializer, which can be used in a variable declaration where the = initializer is permitted. For example:

```
unsigned short          avar @ 0x3FFE;
```

This declaration places an unsigned short variable called "avar" at address 03FFEH. "avar" can be accessed in exactly the same manner as any other variable of the same type, however no storage is reserved for it. The compiler implements absolute variables by defining the appropriate identifier with an EQU psuedo-operation in the code generated. The declaration above would cause the following line to be generated:

```
_avar     equ    03ffeh
```

5. Source Level Debugger

A source level debugger, Lucifer, is included with this release of the compiler. A manual for Lucifer is supplied, see this for more details.

6. New Compiler Options

ZC now recognizes three new compiler options. These are:

-Gfilename

Generates a symbol file for use with the debugger Lucifer. To compile "test.c" with a symbol file "test.sym":

```
ZC -Gtest.sym test.c
```

-Aromaddr,ramaddr,ramsize

Sets the program, data and stack addresses for ROM based programmes where:

romaddr	= the address where the program code starts
ramaddr	= the address where the programs variable space starts.
ramsize	= the number of bytes of combined data and stack space which is available. The stack starts at address ramaddr + ramsize

-CPM

Causes the compiler to generate a CP/M .COM file instead of a .HEX file, and use the CP/M runtime startoff zcrtcpm.obj instead of the ROM runtime startoff zcrt.obj.

7. New Runtime Startoff Modules

The compiler is now supplied with two extra runtime startoff modules, zcrt.obj and zcrti.obj. zcrt.obj is the default ROM runtime startoff module and includes code to setup the RST and interrupt vectors to point into the __vectors table used by set_vector(). zcrt.obj should be used if your program will be placed in RAM or ROM starting at address 0. If your program is to be placed in RAM or ROM at an address other than 0; the alternate ROM runtime startoff zcrti.obj should be used. zcrti.obj is nearly identical to zcrt.obj except it does not contain the vector jump table or the code to setup the interrupt and RST vectors.

8. Bug Fixes in v4.11 of the Z80 Compiler

Version 4.11 of the HI-TECH C Z80 compiler has some bug fixes and new features from version 4.0x:

In the Assembler (ZAS), JR and DJNZ instructions caused jump out of range errors when an absolute origin for the code had been set by the ORG directive.

The Linker (LINK) now supports a -F option to produce a symbol file, i.e. a .OBJ file that contains only symbol records. This is useful when generating overlay code.

The Optimizer (OPTIM) had a bug which caused it to hang or crash on certain unusual code sequences, this has new been fixed.

The Code Generator (CGEN) contained a serious bug which caused bad stack adjustment code to be generated for functions containing more than 12 bytes of local (auto) variables. The stack pointer was added to instead of subtracting, causing corruption of the stack.

In V4.0x, type casts from *float* or *double* to a *char* defined as *extern* caused incorrect conversion code to be generated.

In V4.0x, the compiler driver ZC.EXE passed an incorrect argument to pass 1 of the compiler, resulting in the *port* qualifier not being recognized.

NEW LIBRARY FUNCTION

SET_VECTOR

SYNOPSIS

```
#include        <intrpt.h>
typedef interrupt void (*isr)();
isr set_vector(isr * vector, isr func);
```

DESCRIPTION

This routine allows an interrupt vector to be initialized. The first argument should be the address of the interrupt vector (not the vector number but the actual address) cast to a pointer to *isr*, which is a typedef'd pointer to an interrupt function. The second argument should be the function which you want the interrupt vector to point to. This must be declared using the *interrupt* type qualifier. The return value of *set_vector()* is the previous contents of the vector.

SEE ALSO