HI-TECH C Compiler for PIC10/12/16 MCUs Version 9.80 Release Notes

Produced on: August 16, 2010

Australian Design Centre
45 Colebard Street West
Acacia Ridge QLD 4110
Australia

web: http://www.htsoft.com

THIS FILE CONTAINS IMPORTANT INFORMATION RELATING TO THIS COMPILER. PLEASE READ IT BEFORE RUNNING THIS SOFTWARE.

Chapter 1

Introduction

1.1 Description

This 9.80 version of the compiler is a minor update, which has support for the latest devices and fixes bugs reported in previous versions, particularly with Baseline PIC devices.

The earlier 9.70 compiler release replaced both the previous HI-TECH C Compiler for PIC10/12/16 MCUs (formally the PRO compiler) and the HI-TECH PICC STD Compiler for the Microchip PICmicro. The PICC STD compiler will no longer be developed but will continue as an operating mode of this, and subsequent, compiler releases.

1.2 Updates and Feedback

Microchip welcomes bug reports, suggestions or comments regarding this compiler version. Please direct any bug reports or feature requests via email to (support).

1.3 Previous Versions

The previous version of HI-TECH C Compiler for PIC10/12/16 MCUs was 9.71a, released in June 2010. The previous HI-TECH PICC STD Compiler for the Microchip PICmicro was 9.60PL3, released in January 2009.

Chapter 2

Stop Press

The following are changes made that have not been released in the user's guide.

2.1 C-Assembly Coding

The code example and text in the manual section 3.9.1 is no longer valid as a result of changes in the compiler. Assembly routines cannot define their own local variables. All FN-type directives (e.g. `FNCALL`, `FNSIZE` etc) are no longer supported by the compiler and should not be used.

2.2 Delay Functions

The `__delay_ms` and `__delay_us` function descriptions in the manual indicate that a macro `_XTAL_FREQ` needs to be defined to be the oscillator frequency. This value should be a frequency 4 times the instruction frequency. It may not be the same as the crystal frequency if multipliers are used.

2.3 The default pointer class format

A new feature in 9.71a will force C pointer variables to a known "default" class if they are referenced in assembly. This class is described in the manual section "Pointers to Both Memory Spaces", and the specific encoding of these pointers, which is required if accessing them from assembly, is as below.

For all devices, the size of these pointers is 2 bytes. For Baseline and Mid-range PIC devices, the most significant bit is clear if the address is that of a data memory-based object; set if it is that of an object in program memory. For Enhanced Mid-range PIC devices, the state of this bit is reserved: set indicates a program memory address; clear the address of an object in data memory.

2.4 FN-type Directives

All FN-type directives are no longer support by the compiler for combined C and assembly projects. The example shown in the manual in Section 3.9.1 makes use of the `FNSIZE` directive and references the `FNCALL` directive and so will not compile using the latest compiler release. These directives should not be used and the code adjusted to accordingly. Assembly routines should use global symbols rather than attempt to define their own local (auto) symbols using the `FNSIZE` directive. The compiler will not take into account any stack used by routines called from assembly code.

Section 3.9.4.3 also describes these directives. This section is no longer relevant.

2.5 Messages

The following warning and error messages are new and do not appear in the manual.

(1262) object "*" lies outside available * space (Code Generator)

An absolute variable was positioned at a memory location which is not within the memory defined for the target device, e.g.

```
int data @ 0x800 /* oops -- is this the correct address? */
```

(1273) Omniscient Code Generation not available in Lite mode (Driver)

This message advises that advanced features of the compiler are not be enabled in this Lite mode compiler.

(1275) only functions may be qualified "*" (Parser)

A qualifier which only makes sense when used in a function definition has been used with a variable definition.

```
interrupt int dacResult; /* oops --
the interrupt qualifier can only be used with functions */
```

(1276) buffer overflow in DWARF location list (Cromwell)

A buffer associated with the ELF/DWARF debug file has overflowed. Contact HI-TECH Support with details.

(1278) omitting "*" which does not have a location (Cromwell)

A variable has no storage location listed and will be omitted from the debug output. Contact HI-TECH Support with details.

(1284) malformed mapfile while generating summary: CLASS expected but not found (Driver)

The map file being read to produce a memory summary is malformed. Either the file has been edited or corrupted, or this is a compiler error ? contact HI-TECH Support with details.

(1285) malformed mapfile while generating summary: no name at position * (Driver)

The map file being read to produce a memory summary is malformed. Either the file has been edited or corrupted, or this is a compiler error ? contact HI-TECH Support with details.

(1286) malformed mapfile while generating summary: no link address at position * (Driver)

The map file being read to produce a memory summary is malformed. Either the file has been edited or corrupted, or this is a compiler error ? contact HI-TECH Support with details.

(1287) malformed mapfile while generating summary: no load address at position * (Driver)

The map file being read to produce a memory summary is malformed. Either the file has been edited or corrupted, or this is a compiler error ? contact HI-TECH Support with details.

(1288) malformed mapfile while generating summary: no length at position * (Driver)

The map file being read to produce a memory summary is malformed. Either the file has been edited or corrupted, or this is a compiler error ? contact HI-TECH Support with details.

(1289) line range limit exceeded, debugging may be affected (Cromwell)

A C statement has produced output whose length exceeds a preset limit. This means that debug information produced by cromwell may not be accurate. This warning does not indicate any potential code failure.

(1290) buffer overflow in DWARF debugging information entry (Cromwell)

A variable has no storage location listed and will be omitted from the debug output. Contact HI-TECH Support with details.

(1291) bad ELF string table index (Cromwell)

An ELF file passed to Cromwell is malformed and cannot be used.

(1292) malformed define in .SDB file * (Cromwell)

The named SDB file passed to Cromwell is malformed and cannot be used.

(1293) couldn't find type for "*" in DWARF debugging information entry (Cromwell)

The type of symbol could not be determined from the SDB file passed to Cromwell. Either the file has been edited or corrupted, or this is a compiler error ? contact HI-TECH Support with details.

(1294) there is only one day left until this licence expires (Driver)

The compiler is running as a demo and will be unable to run in PRO mode after the evaluation license has expired in less than one day?s time. After expiration, the compiler can be operated in Lite mode indefinitely, but will produce a larger output binary.

(1295) there are * days left until this licence will expire (Driver)

The compiler is running as a demo and will be unable to run in PRO mode after the evaluation license has expired in the indicated time. After expiration, the compiler can be operated in Lite mode indefinitely, but will produce a larger output binary.

(1296) source file "*" conflicts with "*" (Driver)

The compiler has encountered more than one source file with the same basename. This can only be the case if the files are contained in different directories. As the compiler and IDEs based the names of intermediate files on the basenames of source files, and intermediate files are always stored in the same location, this situation is illegal. Ensure the basename of all source files are unique.

(1297) option * not available in Lite mode (Driver)

Some options are not available when the compiler operates in Lite mode. The options disabled are typically related to how the compiler is executed, e.g. –GETOPTION and –SETOPTION, and do not control compiler features related to code generation.

(1298) use of * outside macros is illegal (Assembler)

Some assembler directives, e.g. EXITM, can only be used inside macro definitions.

(1299) non-standard modifier "*" - use "*" instead (Parser)

A printf placeholder modifier has been used which is non-standard. Use the indicated modifier instead. For example, the standard hh modifier should be used in preference to b to indicate that the value should be printed as a char type.

(1300) maximum number of program classes reached. List may be truncated (Cromwell)

Cromwell is passed a list of class names on the command line. If the number of number of class names passed in is too large, not all will be used and debugging information may be affected.

(1301) invalid ELF section header. Skipping (Cromwell)

Cromwell found an invalid section in an ELF section header. This section will be skipped.

(1302) could not find valid ELF output extension for this device (Cromwell)

The extension could not be for the target device family.

(1303) invalid variable location detected: * - * (Cromwell)

A symbol location could not be determined from the SDB file.

(1304) unknown register name: "*" (Cromwell)

The location for the indicated symbol in the SDB file was a register, but the register name was not recognised.

(1305) inconsistent storage class for variable: "*" (Cromwell)

The storage class for the indicated symbol in the SDB file was not recognised.

(1306) inconsistent size (* vs *) for variable: "*" (Cromwell)

The size of the symbol indicated in the SDB file does not match the size of its type.

(1307) psect * truncated to * bytes (Driver)

The psect representing either the stack or heap could not be made as large as requested and will be truncated to fit the available memory space.

(1308) missing/conflicting interrupts sub-option, defaulting to "*" (Driver)

The suboptions to the –INTERRUPT option are missing or malformed, e.g.

```
    --INTERRUPTS=single,multi
```

Oops, did you mean single-vector or multi-vector interrupts?

(1309) ignoring invalid runtime * sub-option (*) using default (Driver)

The indicated suboption to the –RUNTIME option is malformed, e.g.

```
    --RUNTIME=default,speed:0y1234
```

Oops, that should be 0x1234.

(1310) specified speed (*Hz) exceeds max operating frequency (*Hz), defaulting to *Hz (Driver)

The frequency specified to the speed suboption to –RUNTIME option is too large for the selected device.

```
    --RUNTIME=default,speed:0xffffffff
```

Oops, that value is too large.

(1311) missing configuration setting for config word *, using default (Driver)

The configuration settings for the indicated word have not be supplied in the source code and a default value will be used.

(1312) conflicting runtime perform sub-option and configuration word settings, assuming *Hz (Driver)

The configuration settings and the value specified with the perform suboption of the –RUNTIME options conflict and a default frequency has been selected.

(1313) * sub-options ("*") ignored (Driver)

The argument to a sub-option is not required and will be ignored.

```
    --OUTPUT=intel:8
    Oops, the :8 is not required
```

(1314) illegal action in memory allocation (Code Generator)

This is an internal error. Contact HI-TECH Support with details.

(1315) undefined or empty class used to link psect * (Linker)

The linker was asked to place a psect within the range of addresses specified by a class, but the class was either never defined, or contains no memory ranges.

(1316) attribute "*" ignored (Parser)

An attribute has been encountered that is valid, but which is not implemented by the parser. It will be ignored by the parser and the attribute will have no effect.

(1317) missing argument to attribute "*" (Parser)

An attribute has been encountered that requires an argument, but this is not present.

(1318) invalid argument to attribute "*" (Parser)

An argument to an attribute has been encountered, but it is malformed.

(1319) invalid type "*" for attribute "*" (Parser)

This indicated a bad option passed to the parser. Contact HI-TECH Support with details.

(1320) attribute "*" already exists (Parser)

This indicated the same attribute option being passed to the parser more than once. Contact HI-TECH Support with details.

(1321) bad attribute -T option "%s" (Parser)

The attribute option passed to the parser is malformed. Contact HI-TECH Support with details.

(1322) unknown qualifier "%s" given to -T (Parser)

The qualifier specified in an attribute option is not known. Contact HI-TECH Support with details.

(1323) attribute expected (Parser)

The __attribute__ directive was used but did not specify an attribute type.

```
    int rv (int a) __attribute__(()) /* oops -- what is the attribute? */
```

(1324) qualifier "*" ignored (Parser)

Some qualifiers are valid, but may not be implemented on some compilers or target devices. This warning indicates that the qualifier will be ignored.

(1325) no such CP* register: ($*), select (*) (Code Generator)

A variable has been qualifier as cp0, but no corresponding co-processor register exists at the address specified with the variable.

(1326) "*" qualified variable (*) missing address (Code Generator)

A variable has been qualifier as cp0, but no corresponding co-processor register exists at the address specified with the variable.

```
    cp0 volatile unsigned int mycpvar @ 0x7000; /* oops --
```

did you mean 0x700, try... */

```
    cp0 volatile unsigned int mycpvar @ __REGADDR(7, 0);
```

(1327) interrupt function "*" redefined by "*" (Code Generator)

An interrupt function has been written that is linked to a vector location that already has an interrupt function lined to it.

```
    void interrupt timer1_isr(void) @ TIMER_1_VCTR { ... }
    void interrupt timer2_isr(void) @ TIMER_1_VCTR { ... } /* oops --
```

did you mean that to be TIMER_2_VCTR */

(1328) coprocessor * registers cannot be accessed from * code (Code Generator)

Code in the indicated instruction set has illegally attempted to access the coprocessor registers. Ensure the correct instruction set is used to encode the enclosing function.

(1329) can only modify RAM type interrupt vectors (Code Generator)

The SETVECTOR() macro has been used to attempt to change the interrupt vector table, but this table is in ROM and cannot be changed at runtime.

(1330) only functions or function pointers may have an instruction set architecture qualifier (Code Generator)

An instruction set qualifier has been used with something that does not represent executable code.

```
mips16e int input; /* oops -- you cannot qualify a variable with an instruction
```

(1331) interrupt functions may not be qualified "*" (Code Generator)

A illegal function qualifier has been used with an interrupt function.

```
mips16e void interrupt tisr(void) @ CORE_TIMER_VCTR; /* oops -- you cannot use
```

(1332) invalid qualifier (*) and type combination on "*" (Code Generator)

Some qualified variables must have a specific type or size. A combination has been detected that is not allowed.

```
volatile cp0 int mycpvar @ __REGADDR(7,0); /* oops -- you must use unsigned typ
```

(1333) cannot extend instruction (Assembler)

An attempt was made to extend a MIPS16E instruction where the instruction is nonextensible. This is an internal error. Contact HI-TECH Support with details.

(1334) invalid * register operand (Assembler)

An illegal register was used with an assembly instruction. Either this is an internal error or caused by invalid hand-written assembly code.

```
psect my_text,isa=mips16e,reloc=4
move t0,t1  /* oops -- these registers cannot be used in the 16-bit instruction
```

(1335) instruction "*" is deprecated (Assembler)

An assembly instruction was used that is deprecated.

```
    beql t0,t1,12  /* oops -- this instruction is no longer supported */
```

(1336) a psect may belong to only one ISA (Assembler)

Psects that have a flag that defines the allowed instruction set architecture. A psect has been defined whose ISA flag conflicts with that of another definition for the same psect.

```
    mytext,global,isa=mips32r2,reloc=4,delta=1
    mytext,global,isa=mips16e,reloc=4,delta=1  /* oops -- is this the wrong psect n
```

(1337) instruction/macro "*" is not part of psect ISA (Assembler)

An instruction from one instruction set architecture has been found in a psect whose ISA flag specifies a different architecture type.

```
    psect my_text,isa=mips16e,reloc=4
    mtc0 t0,t1 /* oops -- this is a 32-bit instruction */
```

(1338) operand must be a * bit value (Assembler)

The constant operand to an instruction is too large to fit in the instruction field width.

```
    psect my_text,isa=mips32r2,reloc=4
    li t0,0x123456789 /* oops -- this constant is too large */
```

(1339) operand must be a * bit * value (Assembler)

The constant operand to an instruction is too large to fit in the instruction field width and must have the indicated type.

```
    addiu a3, a3, 0x123456 /* oops -- the constant operand to this MIPS16E instruct
```

(1342) whitespace after "\" (Preprocessor)

Whitespace characters have been found between a backslash and newline characters and will be ignored.

(1343) hexfile data at address 0x* (0x*) overwritten with 0x* (Objtohex)

The indicated address is about to be overwritten by additional data. This would indicate more than one section of code contributing to the same address.

(1346) can't find 0x* words for psect "*" in segment "*" (largest unused contiguous range 0x%lX) (Linker)

See also message (491). The new form of message also indicates the largest free block that the linker could find. Unless there is a single space large enough to accommodate the psect, the linker will issue this message. Often when there is banking or paging involved the largest free space is much smaller than the total amount of space remaining,

(1346) can't find 0x* words (0x* withtotal) for psect "*" in segment "*" (largest unused contiguous range 0x%lX) (Linker)

See also message (593). The new form of message also indicates the largest free block that the linker could find. Unless there is a single space large enough to accommodate the psect, the linker will issue this message. Often when there is banking or paging involved the largest free space is much smaller than the total amount of space remaining,

(1348) enum tag "*" redefined (from *:*) (Parser)

More than one enum tag with the same name has been been defined, The previous definition is indicated in the message.

```
enum VALS { ONE=1, TWO, THREE };
enum VALS { NINE=9, TEN }; /* oops -- is INPUT the right tag name? */
```

(1350) pointer operands to "-" must reference the same array (Code Generator)

If two addresses are subtracted, the addresses must be of the same object to be ANSI compliant.

```
int * ip;
int fred, buf[20];
ip = &buf[0] - &fred; /* oops -- second operand must be an address of a "buf" e
```

(1352) truncation of operand value (0x*) to * bits (Assembler)

The operand to an assembler instruction was too large and was truncated.

```
movlw 0x321 ; oops -- is this the right value?
```

second operand must be an address of a "buf" element */

(1354) ignoring configuration setting for unimplemented word * (Driver)

A configuration word setting was specified for a word that does not exist on the target device.

```
__CONFIG(3, 0x1234);  /* config word 3 does not exist on an 18C801 */
```

(1355) inline delay argument too large (Code Generator)

The inline delay sequence _delay has been used, but the number of instruction cycles requested is too large. Use this routine multiple times to achieve the desired delay length.

```
#include <htc.h>
void main(void) {
  delay(0x400000); /* oops -- cannot delay by this number of cycles */
}
```

(1356) fixup overflow referencing * * (0x*) into * byte* at 0x*/0x* -> 0x* (* */0x*) (Linker)**

See also message (477). This form of the message precalculates the address of the offending instruction taking into account the delta value of the psect which contains the instruction.

(1357) fixup overflow storing 0x* in * byte* at 0x*/0x* -> 0x* (* */0x*) (Linker)**

See also message (477). This form of the message precalculates the address of the offending instruction taking into account the delta value of the psect which contains the instruction.

(1358) no space for * temps (*) (Code Generator)

The code generator was unable to find a space large enough to hold the temporary variables (scratch variables) for this program.

(1359) no space for * parameters (Code Generator)

The code generator was unable to find a space large enough to hold the parameter variables for a particular function.

(1360) no space for auto/param * (Code Generator)

The code generator was unable to find a space large enough to hold the auto variables for a particular function. Some parameters passed in registers may need to be allocated space in this auto area as well.

(1361) syntax error in configuration argument (Parser)

The argument to #pragma config was malformed.

```
#pragma config WDT /* oops -- is WDT on or off? */
```

(1362) configuration setting *=* redefined (Code Generator)

The same config pragma setting have been issued more than once with different values.

```
#pragma config WDT=OFF
#pragma config WDT=ON /* oops -- is WDT on or off? */
```

(1363) unknown configuration setting (* = *) used (Driver)

The configuration value and setting is not known for the target device.

```
#pragma config WDR=ON /* oops -- did you mean WDT? */
```

(1364) can’t open configuration registers data file * (Driver)

The file containing value configuration settings could not be found.

(1365) missing argument to pragma "varlocate" (Parser) The argument to #pragma varlocate was malformed.

```
#pragma varlocate /* oops -- what do you want to locate & where? */
```

(1366) syntax error in pragma "varlocate" (Parser)

The argument to #pragma varlocate was malformed.

```
#pragma varlocate fred /* oops -- which bank for fred? */
```

(1367) end of file in _asm (Parser)

An end-of-file marker was encountered inside a _asm _endasm block.

(1368) assembler message: * (Assembler)

Displayed is an assembler advisory message produced by the MESSG directive contained in the assembler source.

(1369) can’t open proc file * (Driver)

The proc file for the selected device could not be opened.

(1371) float type can’t be bigger then double type; double has been changed to * bits (Driver)

The size of a double type cannot be smaller than that of a float type. Adjust the –FLOAT and –DOUBLE options accordingly.

(1372) interrupt level cannot be greater than * (Code Generator)

If using the #pragma interrupt_level, the specified level must be lower than that indicated in the error.

(1372) interrupt level cannot be greater than * (Code Generator)

If using the #pragma interrupt_level, the specified level must be lower than that indicated in the error.

(1375) multiple interrupt functions (* and *) defined for device with only one interrupt vector (Code Generator)

The selected device has one interrupt vector, and as such, can have one interrupt function defined. The compiler has encountered more than one such function in the input source.

(1376) initial value (*) too large for bitfield width (*) (Code Generator)

A structure with bitfields has been initialized; however, one of the initial values is too large to fit in the corresponding bitfield member. Either the bitfield size is too small, or the value is too large.

(1377) missing argument to pragma "*" (Parser)

A pragma has been used that needs an additional argument. For example the #pragma addrqual must be followed by an argument such as ignore, request, require or reject.

```
#pragma addrqual   /* oops -- what is the selection? */
```

(1378) syntax error in pragma "*" (Parser)

The argument to a pragma is not one expected.

```
#pragma addrqual forget  /* oops -- not a valid selection */
```

(1379) no suitable strategy for this switch (Code Generator)

The compiler uses one of several different strategies to generate code associated with switch statements and the corresponding cases. The compiler was not able to determine which strategy to use.

(1380) unable to use switch strategy "*" (Code Generator)

The #pragma switch may be used to indicate a preference for the switch code strategy. The selection made cannot be accommodated by the compiler.

(1387) inline delay argument must be constant (Code Generator)

The argument to the _delay pseudo function which inserts an in-line delay, must be a constant expression.

```
_delay(delayVal); /* oops -- argument must be a constant */
```

(1390) identifier specifies insignificant characters beyond maximum identifier length (Parser)

An identifier (symbol) whose length exceeds the current maximum identifier length. Characters after this maximum are ignored. Either use shorter names or increase the identifier length via the `-N` option.

```
int aVariableWithAnExtremelyLongIdentifier;
/* oops -- this exceeds the default identifier length */
```

Chapter 3

New Features

The following are new features the compiler now supports. The version number in brackets indicates the first compiler version to support the feature.

3.1 General

New devices supported (9.71a) The following devices are now supported by the compiler: 12F1840, 16F1616, 16F1617, 16F1618, 16F1619, 16F1626, 16F1627, 16F1824, 16F1825, 16F1828, 16F1829, 16F1999, 16F720, 16F721, 16F722A, 16F723A, 16LF1902, 16LF1903. Support has also been added for the "LF" versions of these parts were they exist.

New devices supported (9.70) The following devices are now supported by the compiler: 12F520, 12F1822, 12LF1822, 12F617, 16F1823, 16LF1823, 16F1938, 16LF1938, 16F1939, 16LF1939, 16F1946, 16LF1946, 16F1947, 16LF1947, 16F707 and 16LF707.

Standard operating mode (9.70) This compiler can operate in one of three mode: PRO, Standard or Lite. PRO and Standard modes require a license, but may be evaluated for 45 days free of charge. Lite mode is always available. The difference in operating mode is purely one of output code size — all source code features and code generation options are implemented in all modes. Providing the license allows, the compiler may be switched from one mode to another on a build-by-build basis.

3.2 Code Generator/Parser

Bank selection after in-line assembly code (9.80) When the compiler sees inline asm, it will now assume that this code has affected the state of the currently selected bank.

Anonymous structure members (9.71a) Added support for members of anonymous structures to have the same name provided they are of the same type, and are at the same offset within the outermost union.

Address qualifier pragma (9.71a) A addrqual pragma has been introduced to allow control over the meaning of various variable qualifiers. The options and meaning are the same as for the existing `--ADDRQUAL` option, e.g. `#pragma addrqual request`. Only one selection may be made using this pragma.

Longer inline delays (9.71a) Longer delays are possible using the in-line delay feature of the compiler. Three nested loops can now be employed when using the `_delay` pseudo routine.

Pointers accessed from assembly code (9.71a) If any C pointer variables are accessed in separate assembly modules, the pointers will now take on a fixed format. Their size and format will be made such that the pointer can access any object in any memory space. This will ensure that the code will function correctly, but will mean that pointers may be larger than necessary and code to manipulate or dereference the pointers will be larger and slower.
The pointer's size will be 2 bytes. The pointer will contain the full address of the target object, however the MSb is used to indicate the memory space in which the target resides. On Mid-range and Baseline devices, the MSb is set to indicate the target address is in the data memory, and is clear to indicate the address is that of an object in program memory. On Enhanced Mid-range devices, the MSb has the opposite state: set implies program memory; clear implies data memory.

Updated Call Graph (9.71a) The code generator produces the call graph, and other diagnostic information, which will be shown in the assembly list file. The information in the call graph tables has been expanded to contain more information. In addition, a full call graph listing, in a style similar to those that appeared in the map files prior to version 9.70, is also printed. This latter is the same information, but presented in a different form. A printout of the critical paths in the call graph is now also displayed.

Absolute addressed const objects (9.70) Both initialized and uninitialized `const`-qualified objects can now be placed at an absolute address. This feature allows users to position data in program memory at a specific locations with out the need for assembly code. Examples:

```
const unsigned short foo[] @ 0x100 = { 0xADDE, 0xEFBE, 0xEDFE };
const int bar @ 0x200 = 0x0DD0;
```

Functions larger than a page (9.70) The compiler can now compile functions whose assembly output is larger than a ROM page size. This is the first mid-range compiler that has this limitation lifted. No source code changes are required for this to take place. This feature is not available for baseline devices.

Improved memory allocation for local objects (9.70) The memory allocation strategy for local objects: autos, parameters and temporary variables, has been improved. Previously a compiled stack was formed from the auto-parameter blocks of each function in the program and was allocated one contiguous range of memory that had to be located within one RAM bank. This limitation has been lifted. The compiled stack can now consist of smaller stacks, each located in separate RAM banks. The RAM bank chosen for a function's auto variables may be different to that used by its parameters, and indeed the auto variables associated with one function may even be spread over multiple RAM banks, if required. The parameters and temporary variables associated with a function will only be allocated to one RAM bank, but those of other functions may use different banks.

RAM objects larger than one bank (9.70) When the target device is an enhanced mid-range PIC device, the source code may define RAM-based objects (arrays and structures) that are larger than one bank in size. Such objects will be accessed indirectly using the linear data memory. This feature is not available when a baseline or mid-range target device is selected.

More than one bank of initialized objects (9.70) Previously the total size of initialized objects defined by a program must be less than the size of a RAM bank. This limitation has been lifted. Initialized objects can reside in more than one bank, and the automatically generated runtime startup code can ensure that all areas are correctly initialized.

Psect pragma reinstated (9.70) The `#pragma psect` directive, which was originally present on STD compilers, was initially not available on PRO compilers. This directive has now been reinstated and operates as it did previously. Although most objects can now be placed at specific locations by making them absolute, the psect pragma permits more control over how objects are positioned by allowing use of the many features in the linker.

3.3 Assembler

Optimization controls (9.71a) New assembler controls have been added which allows the user to specify sections of assembly code that should not be optimized. The controls are `OPT asmopt_on` and `OPT asmopt_off` which enable and disable, respectively, the assembler optimizer when encountered.

3.4 MPLAB Plugin

Plugin version 1.35 (9.70) The Windows version of this release features a new version of the HI-TECH Universal Toolsuite MPLAB plugin, viz. version 1.35. Only this version of the plugin is compatible with HI-TECH C Compiler for PIC10/12/16 MCUs 9.70. The plugin is installed automatically when the compiler is installed. The changes to the plugin reflect changes to the driver options in 9.70.

Chapter 4

Changes

The following are features that are now handled differently by the compiler. These changes may require modification to your source code if porting code to this compiler version. The version number in brackets indicates the first compiler version to implement the change.

4.1 General

Stack Call Depth in List File (9.71a) The stack depth indicated for each function in the Call Graph Tables now indicates the maximum possible depth at which that function can appear in the program's call graph. A function may appear in several places in the call graph, and it is always the maximum depth that is shown. These tables are shown in the assembly list file.

4.2 Driver

–apbank option (9.70) This option is no longer applicable. If used the driver will ignore it.

Linker option overrides (9.70) The operation of the `-L-` driver option, which passes options directly to the linker, has changed. If this option is being used to add additional linker options, then there is no change required in the way it is used. If this option is being used to replace default linker options then there may be changes required to the driver options. Previously if the string up to the first equal sign matched the *initial* part of a default linker option, that entire linker option would have been removed and what follows the `-L-` would be added as a new option. Now a search is now made for the string (up to the first equal sign in the `-L-` option) and each and every psect or class name in the default linker options. If a match is found, that psect/class specification and the remaining part of the linker option is removed and the new option added.

Special pointer targets (9.70) Pointers and integers are not interchangeable. Assigning an integer constant to a pointer will generate a warning to this effect. There is no information in the integer constant relating to the type, size or memory location of the destination. There is a very good chance of code failure if pointers are assigned integer addresses and dereferenced, particularly for devices like PIC devices which have more than one address space. For information about this issue and ways of dealing with it, refer to *Special Pointer Targets* in the user manual.

4.3 Code Generator

Pointer format (9.71a) The meaning of the most significant bit for a 2-byte pointer has been reversed. Now, when set, this indicates that the pointer contains the address of an object in the program memory space; clear for objects in the data memory. This has been changed in the Stop Press section of these release notes.

Stack management (9.71a) The way the hardware stack is used by a program has changed. Previously mid-range devices would always use the hardware stack to store function return addresses. Baseline devices would also use the stack, but, as their stack is very limited, the compiler would revert to a lookup table convention to allow a call depth larger than the stack would normally allow. The changes are summarized below.

- All devices by default only use the available hardware stack. If this depth is exceeded, a warning will be issued by the compiler. The `stackwarn` suboption to the `--RUNTIME` option is no longer available.
- If the `--RUNTIME` suboption `stackcall` is used, for all devices the compiler will change to used a lookup table calling convention if the hardware stack level is exceeded.
- The `fastcall` function qualifier is no longer implemented. The calling method is determined purely by the compiler option.

Switch code strategies (9.71a) Changes have been made to the way the compiler produced code associated with `switch` statements. Most changes are internal, but the pragma that allows the user to specify the switch type used has also changed. The `#pragma switch` now takes the arguments `speed`, `space` and `time`, which direct the compiler to choose the fastest switch code, the smallest switch code, or switch code that results in a constant time delay for each case, respectively. The old pragma arguments, e.g. `direct` and `simple` are now deprecated. If no pragma is used in the code, the compiler make a choice based on the case values and on the global optimization setting.

Pointers accessed from assembly code (9.71a) If any C pointer variables are accessed in separate assembly modules, the pointers will now take on a fixed format. Their size and format will be made such that the pointer can access any object in any memory space. This will ensure that the code will function correctly, but will mean that pointers may be larger than necessary and code to manipulate or dereference the pointers will be larger and slower.

The pointer's size will be 2 bytes. The pointer will contain the full address of the target object, however the MSb is used to indicate the memory space in which the target resides. On Mid-range and Baseline devices, the MSb is set to indicate the target address is in the data memory, and is clear to indicate the address is that of an object in program memory. On Enhanced Mid-range devices, the MSb has the opposite state: set implies program memory; clear implies data memory.

Assembly names of auto symbols (9.70) Previously `auto` variables associated with a function could be accessed in assembly code either by a linker-defined symbol of the form `??_`*`funcName`*`+`*`offset`* (where *`funcName`* is the name of the function which defines the auto, and *`offset`* is a numerical offset from this symbol) or by a code-generator-defined symbol of the form `funcName_varName`. The first form can no longer be used. Note also that in the latter form, the *underscore* character has been replaced with an `@` symbol, so for example an `auto` variable called `buf` defined in the `main` function could be accessed in assembly code using the symbol `main@buf`. Parameters and temporary variables can be accessed using either form.

Assembly names of static functions and variables (9.70) Previously `static` symbols had assembly domain symbols that were based on an "F" symbol, e.g. `_add_F4035`. Now all assembler domain `static` symbols follow a similar convention to that for other locals. The assembly name of `static` variables are of the form `funcName@varName` where *`funcName`* is the name of the function that defines the symbol. If the variable is not defined inside a function, the symbol has the form: *`fileName`*`@`*`varName`*, where *`fileName`* is the name of the source file that contains the definition. If there are two `static` functions with the same name and each contain a `static` variable of the same name, then the variables will be accessible using the form: *`fileName`*`@`*`funcName`*`@`*`varName`*.

#pragma interrupt_level (9.70) The automatic duplication of functions can be inhibited by preceding a function definition with the interrupt_level pragma. In addition to preventing the function from being duplicated, all functions that it calls, and all functions called by those calls, etc... will also be prevented from being duplicated.

4.4 Assembler

Use of FN-type Directives in Assembly (9.71a) The compiler no longer supports the use of any FN-type assembler directive in projects which consist of both C and assembly code. This includes the `FNCALL` and `FNSIZE` assembler directives. This will mean that the assembly codes contribution to the call stack will not be known by the compiler. Assembly routines also cannot reserve memory for their own auto or parameter symbols and must use global symbols. The example in the manual section 3.9.1 is no longer valid as a result of this change.

Optimization of in-line assembly (9.71a) Previously all assembly code would be optimized if the assembler optimizer was enabled. This is not longer the case. Assembly code that is added in-line with C code by the programmer will not be optimized by the assembler.

goto instruction (9.71a) The assembler will automatically insert instructions for adjusting PCLATH appropriately so that the goto instruction can correctly reach the destination label/address. The assembler will not perform this operation on inline assembler or user-supplied assembly modules.

4.5 Library

Floating-point arithmetic functions (9.70) The compiler library functions which provide 24- and 32-bit arithmetic routines have been updated so that they require less memory and have faster execution times.

Chapter 5

Limitations

The following are limitations in the compiler's operation. These may be general coding restrictions, or deviations from information contained in the user's manual.

5.1 General

Memory Report for Absolute Variables If the user defines absolute variables (variables placed at an absolute address via the `@` construct), and any of these variables overlap other variables in memory, the compiler will also count overlapping variables when it comes to reporting on memory usage. That is, if two, byte-sized variables are located at the same address, the report will indicate 2 bytes being used, rather than 1.

Sample Code The same code provided with the compiler is for reference only. It may not be appropriate for all devices and situations. Use this code as the basis for your own programs, however ensure that it is reviewed before compiling.

5.2 MPLAB IDE

Out of bound variables in watch view If variables are defined for Enhanced Mid-range devices that are large and placed in the linear memory space, MPLAB IDE shows these objects are being "out of bound". This is display issue and is not indicative of code failure.

Wrong Watch Values The IDE expects debug information to be available in COFF. This file format does not allow the compiler to specify the necessary information for the IDE to be able to correctly interpret some pointer variables. The compiler allocates pointer sizes based on their usage, which can make this issue seem "hit and miss". Any attempt to examine pointers in the IDE's Watch view may result in incorrect values being shown, or an "invalid" message being displayed in the view. These issues will correct themselves when ELF is employed to convey debug information in future IDE/compiler versions.

5.3 C Code

Main function size If the function `main` produces assembly code that is 0x1FF words long, this cannot be placed in the program memory and a "can't find space" error message is produced. Increasing or decreasing the size of the function will allow it to be positioned correctly and the error will not be displayed. This only affects Baseline PIC devices.

Ragged Arrays inside Structure When an array of pointers to const objects is made part of a structure, the size of the pointers used to access each element of the array may have the wrong size and result in erroneous access to the elements.

Functions Called from In-line Assembly The code generator will not be able to identify a C function called only from in-line assembly code if the definition for that C function is placed before the assembly call instruction in the source file. Placing the function definition after the call is acceptable. If the function cannot be identified, no code will be generated for the function and the linker will issue undefined symbol errors.

Can't Generate Code Messages When compiling for baseline devices, some complex expressions may cause compile-time errors `(712) Can't generate code for this expression`. The expressions should be simplified to work around this. This may require the use of additional variables to store intermediate results. This is most likely with long integer or floating-point arithmetic.

Indirect function calls A function should not be called indirectly (via a function pointer) from both main-line code and interrupt code. In cases where the function would be duplicated, this may result in incorrect operation of the code. A function may be called directly from both main-line and interrupt code, or indirectly from either call graph.

eeprom qualifier This qualifier is not available and its use will result in undefined behaviour. This qualifier is planned for a subsequent release and information in the manual regarding this qualifier should be disregarded until that time.

option and tris For baseline devices, the OPTION and TRIS registers must be read/written as a byte. Reading or writing individual bits is not supported.

PIC17 support PIC 17 devices (for example, 17C756) is not supported by this compiler.

fast32 floats The option to select the fast 32-bit float or double library for PIC17 devices that was included in the PICC STD compiler is no longer available.

5.4 Assembler

Procedureal Abstraction An instance has been seen, but not resolved, which appears to be related to the proceedural abstraction optimization of the assembler. If suspect behaviour is encountered, compiling for speed, instead of space, will disable this optimization and allow you to confirm if this is the underlying cause. This problem was reported for an Enhanced Mid-range device.

5.5 Libraries

Flash Read for 12F617 The FLASH_READ function cannot be used with this device as the routine uses different registers to those available on this device.

Peripheral routines The flash read/write routines/macro included with this compiler are functional only for devices that can write one word at a time (e.g. 16F877).

5.6 Header Files

Register Names for 12F617 Some names defined in the device header file differ from the name specified in the device datasheet.

- INTCON bits: TMR0IE and TMR0IF in the header file represents T0IE and T0IF in the datasheet
- PIR1 bits: CCP1IF represents ECCPIF, C1IF represents CMIF
- VRCON bits: C1VREN represents CMVREN
- CMCON0 bits: C1ON, C1OUT, C1OE, C1POL, C1R represents CMON, COUT, CMOE, CMPOL, CMR, respectively
- CMCON1 bits: C1HYS, C1SYNC represents CMHYS, CMSYNC, respectively
- PIE1 bits: ECCPIE, C1IE represents CCP1IE, CMIE, respectively
- PMADRL bits: PMADRL0, PMADRL1, PMADRL2, PMADRL3, PMADRL4, PMADRL5, PMADRL6, PMADRL7 are missing from the header file

Register Names for 16(L)F1946/47 Some names defined in the device header file differ from the name specified in the device datasheet.

- OPTION_REG bits: TOCS and TOSE in the datasheet are represented as TMROSE and TMROCS in header file

Register Names for 16(L)F1933/34/36/37/38/39 Some names defined in the device header file differ from the name specified in the device datasheet.

- Shadow STSTUD register bits: C, DC and Z in the datasheet are represented as C_SHAD, DC_SHAD and Z_SHAD in header file
- LCD segment registers: The SEx bits of the segments registers are represented by the symbols SEGx in the header files.

Configuration Bits for 16(L)F1936/37 The DEBUG configuration bit is missing from the device header files.

Chapter 6

Bug Fixes

The following are corrections that have been made to the compiler. These may fix bugs in the generated code or alter the operation of the compiler to that which was intended or specified by the user's manual. The version number in brackets indicates the first compiler version to implement the fix.

6.1 Preprocessor

Crash when building in MPLAB IDE (9.71a) MPLAB IDE uses the dependency checking feature of the preprocessor when doing incremental builds. If an error was encountered in the project when building, this could have resulted in the preprocessor crashing. This situation has been corrected; any errors detected will result in MPLAB IDE performing a full rebuild.

6.2 Code generator/Parser

Stack overflow (9.80) The code generator didn't take into account some levels of hardware stack required below a function when calculating the 'opt stack' control value for each function. As a result the assembler would perform procedural abstraction where there could potentially be no levels of stack available at runtime and a stack overflow result.

Bad code involving shifts (9.80) Code that involves shifts of byte-sized objects may have destroyed the contents of WREG and resulted in incorrect results. This has been corrected.

Use of temporary variable and interrupts (9.80) A temporary variable (`btemp`) was being used by complex statements in main-line code. This variables was also used in interrupt routines which could result in its contents being destroyed (and code failure) if an interrupt occurred. Such variables are no longer used by main-line code.

Structures containing pointer arrays (9.80) The storage size for pointers in an array that is itself a structure member may not have be correct. This would then result in errors when the pointers were accessed. Pointer size is now correctly determined.

Better debug information (9.71a) For trivial loops, such as `while(1)`, better debug information is produced by the compiler that should result in better source-level debugging in the IDE.

Syntax error (9.71a) This error may be issued for code involving addition. A malformed assembly instruction was being produced by the code generator and being detected by the assembler.

Address of absolute bits (9.71a) The address allocated to absolute bit variables may be incorrect. This would not affect ordinary bit variables defined.

Better memory allocation (9.71a) Better memory allocation of data objects should result in less can't find space errors, particularly when dealing with large objects defined in the code, such as arrays.

Bad behaviour with pointers to structure members (9.71a) In some cases, pointers that are assigned the address of members of a structure may trigger bad behaviour, such as can't generate code messages, or incorrect operation. This will only affect complex assignments such as `&sptr->array[2].`

Incorrect bank selection after shift (9.71a) If shifting by a variable shift count that happens to contain zero, the incorrect bank may be used after the shift if the Bank of the shift count and the bank of the quantity being shifted are different.

Spurious warning regarding arithmetic overflow (9.71a) In some instances the warning relating to arithmetic overflow in constant expression was issued for code that did not appear to suffer from such a condition. This warning was triggered after the code was modified internally, but the warning has now been suppressed in such situations.

Can't generate code comparing bytes (9.71a) Code which compared signed bytes, e.g. `a <= b` in some cases may have triggered this error.

Can't generate code with long to float conversion (9.71a) Code requiring conversion of long to float types, as in code such as `long += float`, may produce this error.

Incorrect results with AND equals (9.71a) Code implementing &= (which may include seemingly unrelated operators, such as %=) may produce bad results.

Incorrect pointer decrement (9.71a) Decrementing a pointer by 1 may not yield the correct result if the pointer is two bytes wide.

Wrong results using compound assignment (9.71a) Compound assignments, such as `/=` and `%=` etc, that had signed operands on the right hand side were not promoting these operands correctly and may have resulted in incorrect results.

Incorrect switch code with 3-byte types (9.71a) Using switch statements with a `short long` type would not work correctly. Switches have been expanded to allow this functionality.

Crash when calling function indirectly (9.71a) A function called indirectly using a pointer that is uninitialized may result in the code generator crashing.

Can't generate code (9.71a) Operators like `*=` with operands that both required the use of the FSR register would result in a register conflict and this error message.

Parameter overwrite (9.71a) Calling a function where one or more arguments to the function were evaluated by calling another function which also called the first function, resulted in the parameter being overwritten. This persistent issue has finally been resolved.

Can't generate code in convention (9.71a) This error may have resulted if code required a conversion from short long to a 16-bit size.

Incorrect subtraction of 3-byte int (9.71a) Three byte subtraction may fail in instances where carry (borrow) should be propagated to higher order bytes.

Overwritten Initialized variable (9.71a) In some instances, an initialized variable's value may be overwritten by the runtime startup code. This would only affect devices with common memory.

Decrement of structure member (9.71a) In some instances in PRO mode only, use of the decrement operator with structure members may produce an incorrect result.

Can't generate code for shifts (9.71a) An error may have resulted if trying to shift an amount specified by the negative of a variable, e.g. `i >> (-k)`

Crash when duplicating functions (9.71a) The code generator may crash in some instances where there are functions called from main-line and interrupt code and which are duplicated.

No space errors (9.71a) In some instances the compiler would indicate that no memory could be found for an object, when there was indeed suitable memory available. A more intelligent search is made of the remaining memory spaces.

Undefined stringtab symbol (9.71a) Some code sequences may have used a `stringtab` symbol, but did not output the code associated with this symbol. This have now been fixed.

Can’t Generate Code for linear accessed arrays (9.71a) The compiler may have produced a Can’t generate code message for code which accessed the first element of arrays accessed linearly on enhanced mid-range PIC devices. This would not affect arrays indexed with a variable, or any code for any other device.

Duplication of undefined functions (9.71a) For functions not defined in the C source (defined by assembly code) it will not be duplicated in situations where this might normally be done.

Bad code for byte operations (9.71a) In some instances when a commutative operator (such as +, &, | etc) had a constant operand, the result would be incorrect. This would only affect byte operations where the other operand was stored in WREG.

Switch code failure (9.71a) The assembly code produced by the code generator for `switch` statements when the `simple` switch type is selected may have resulted in the assembler performing optimizations on the code that would cause code failure. The generated code has been changed to ensure that this cannot happen.

RAM allocation (9.71a) Numerous fixes and changes have been made that fix issues associated with allocation of local RAM variables. In some instances addresses assigned to local objects may be incorrect. These changes also fix instances where there was an over allocation of RAM for local objects. These changes may result in a reduced RAM usage for some projects.

Incorrect overflow warning (9.71a) A warning regarding arithmetic overflow which would sometimes be produced in error is now only issued when the overflow condition does occur.

Bitfield structure initialization (9.71a) Where bitfield structures are defined and initialized, and the number of initial values is less than the number of structure members, the block of initial values contained additional values that would result in the structure being wrongly initialized. This has now been corrected.

Undefined symbol fp_0 (9.71a) The error message `Undefined symbol fp_0` might have been issued if there is function pointers in the code.

Warning 1352 truncation of operand (9.71a) When adding a 16-bit constant and a byte, the generated code did not mask the MSB of the constant and this warning would have been issued. This has been corrected.

Pointer list and call graph truncated (9.71a) In large projects the pointer reference graph and the call graph might have been truncated in the assembler list file.

No space for function parameters (9.71a) The error message `No space for function parameters` might have been issued even if the function had no parameters.

Code generator crashes (9.71a) The code generator might have crashed when compiling projects that contained an interrupt function.

Can’t generate code (9.71a) A subtraction of two bits variable or a constant and a bit variable caused a `can't generate code` error.

6.3 Assembler

Bad optimizations involving PAGESEL (9.80) The assembler optimizer was not correctly identifying the PAGESEL pseudo-op as an instruction that could change the PCLATH register. This may have lead to bad optimizations.

Assembler hangs (9.80) The assembly may have become stuck in an endless loop when processing psects which are absolute (use the `abs` flag). This was caused by two opposing optimizations which undid the work of the other. These optimizations will no longer operate on absolute psects.

Optimizer destroys WREG contents (9.80) In some instances, the assembler optimizer may not see that a CALLW instruction (and the code it calls) destroys the contents of WREG, and results in a previous load of WREG being destroyed. This has now been corrected.

Assembler enters endless loop (9.71a) In some cases the assembler would enter into an endless loop when optimizing code. This could be triggered by assembly code that "jumps to itself".

Removal of subtraction (9.71a) In some situations when `WREG` was known to contain 0, a `SUBWF` instruction may be have been incorrectly removed. If subsequent code used the carry bit, this removal should not have taken place. This has now been corrected and the subtraction left in place.

List file comment truncation (9.71a) Long comment blocks output to the assembly list file may have been truncated. The way that comments are printed has been changed so this will no longer happen and all diagnostic information produced by the code generator will be present in this file.

Optimization of interrupt code (9.71a) The assembler optimizer was not optimizing assembly code produce for interrupt functions for Enhanced Mid-range PIC devices. Interrupt code would work correctly, but would not be optimal. This has been corrected. Interrupt code for other devices was correctly being optimized.

Assembler crash (9.71a) The assembler might have crashed if you used the compiler in PRO or Standard mode and the target device was an Enhanced Mid-range PIC. This would only occur when there was absolute `const`-qualified objects define in the project.

Compilation never ends (9.71a) In some instances the code `while(1);` would cause the assembler optimiser to get stuck in a loop.

Assembler crashes (9.70PL1) The assembler might have crashed if you have an assembler module that defines a macro with a bit instruction.

6.4 Driver

Bad memory ranges (9.80) Where absolute ROM memory ranges (i.e. ranges which did not add to, or subtract from, the default ranges) were correctly specified by the programmer (including page boundaries), these ranges may have been concatenated in such a way that page boundaries relevant to the target device where removed. This may have resulted in code that crossed these boundaries and either compile- or runtime-errors. The page boundaries are now correctly preserved. NB The programmer must specify any memory page boundaries that are relevant to the target device when using this option.

Successful build for code with errors (9.80) If the parser's exit status reported an error and the `--pass1` option was is in effect, the residual p1 file was not being deleted. Build systems (e.g. make, MPLAB IDE) which rely on date-time stamps were seeing the existence of this file to indicate that the preliminary build step was successful. Now, the file is deleted and IDEs forced to rebuild the intermediate p1 file.

until the error in the source C file is resolved.

Crash on error (9.71a) In some situations when the compiler issues an error message, the cleanup process would try to delete a file whose name had not been correctly initialized. This resulted in an exception.

Optimization of runtime startup code (9.71a) The runtime startup module was being treated as an assembly source file and hence not optimized. Optimizations are now performed on this code.

Empty debugger option (9.71a) If the `--DEBUGGER` option was used with now debugger specified after the = character, the driver would crash.

osccal runtime suboption (9.71a) The `osccal` suboption to the `--RUNTIME` option was not being correctly read and disabling this option did not disable this feature. This suboption should now work as expected.

Stack overflow (9.71a) The procedural abstraction optimiser may have replaced two bank selection instructions with a call even if there was no stack space available. This has been fixed.

6.5 Linker and Utilities

Cromwell crash (9.80) In some instances where structures were used before their definition, the resulting diagnostic files may have caused the cormwell application to crash. This situation has been resolved.

Linking of zero-sized psects (9.71a) It was possible an error message may have be issued by the linker if trying to position a psect with zero size, but which specifies a reloc value. The error message will no longer occur and the psect will be linked to default location, however as it is of zero size, this location is not important.

6.6 Libraries and Header Files

Missing ltoa library function (9.80) The `ltoa` library function was not compiled into the supplied libraries, resulting in errors when trying to use this routine. The source code is now built into the libraries.

SFR changes (9.71a) A number of changes have been made to header files as a result of updated information contained in the database from which these files are created. Changes include renaming of SFRs, additional registers being specified or address of registers being corrected. Devices affected are the 12F1822, 12LF1822, 12F617; 16F and 16LF versions of the: 1823, 1826, 1827, 1833, 1834, 1836, 1837, 1838, 1839, 1846, 1847; as well as the 16F684 and 16F707.

Powerdown and timeout bit (9.71a) The address of the powerdown and timeout bits were incorrect. Using these bits may have resulted in a symbol redefinition error.