SDCC Compiler User Guide

16th July 2001

Contents

10(SDCC56)]TJ/F61 9.963 Tf -18.68 0 TdSDCC56

The compiler also allows *inline assembler code* to be embedded anywhere in a function. In addition, routines developed in assembly can also be called.

SDCC also provides an option (–cyclomatic) to report the relative complexity of a function. These functions can then be further optimized, or hand coded in assembly if needed.

SDCC also comes with a companion source level debugger SDCDB, the debugger currently uses ucSim a freeware simulator for 8051 and other micro-controllers.

The latest version can be downloaded from `http://sdcc.sourceforge.net/`.

1.2 Open Source

All packages used in this compiler system are *opensource* and *freeware*; source code

2 Installation

2.1 Linux/Unix Installation

1. Download the source package, it will be named something like sdcc-2.x.x.tg
2. Bring up a command line terminal, such as xterm.
3. Unpack the file using a command like: **"tar -xzf sdcc-2.x.x.tgz**", this will create a sub-directory called sdcc with all of the sources.
4. Change directory into the main SDCC directory, for example type: **"cd sdcc**".
5. Type **"./configure**". This configures the package for compilation on your system.
6. Type **"make**". All of the source packages will compile, this can take a while.
7. Type **"make install"** as root. This copies the bina files, the libraries and the documentation to the install directories.

2.2 Windows Installation

<pending: is this complete? where is borland, mingw>

and header files to /usr/local/share/sdcc/lib and /usr/local/share/sdcc/include.

2.5 Additional Information for Windows Users

<pending: is this up to date?>

as-z80**,**

2.8.5 sdcdb - Source Level Debugger

Sdcdb is the companion source level debugger. The current version of the debugger uses Daniel's Simulator S51, but can be easily changed to use other simulators.

3 Using SDCC

3.1 Compiling

3.1.1 Single Source File Projects

For single source file 8051 projects the process is very simple. Compile your programs **"sdccsourcefile.c".** This will compile, assemble and link your source file. Output are as follows

sourcefile.asm - Assembler source file created by the compiler
sourcefile.lst - Assembler listing file created by the Assembler
sourcefile.rst - Assembler listing file updated with linkedit information, created by linkage editor
sourcefile.sym - symbol listing for the sourcefile, created by the assembler
sourcefile.rel - Object file created by the assembler, input to Linkage editor
sourcefile.map - The memory map for the load module, created by the Linker
sourcefile.ihfi The690The loadfi format the Motorola S19 format with –out-fmt-s19)
sourcefile.cdb - An optional (with –debug) containing debug information

3.1.2 Projects with Multiple Source Files

SDCC can only file at time. us forfiamha(the)-tkilectdeb230(v))

–code-loc<Value> The start location of the code segment, default value 0. Note when this option is used the interrupt vector table is also relocated to the given address. The value entered can be in Hexadecimal or Decimal format, e.g.: –code-loc 0x8000 or –code-loc 32768.

–stack-loc<Value> The initial value of the stack pointer. The default value of the stack pointer is 0x07 if only register bank 0nly re0n6, if other register bank1(re)15are

-E Run only the C preprocessor. Preprocess all the C source files specified

–int-long-reent Integer (16 bit) and long (32 bit) libraries have been compiled as reentrant. Note by default these libraries are compiled as non-reentrant. See section Installation for more details.

–cyclo7atic

3.3.4 bit

This is a data-type and a storage class specifier. When a variable is declared as a bit, it is allocated into the bit addressable memory of 8051, e.g.:

```
bit iFlag;
```

3.3.5 sfr / sbit

Like the bit keyword, *sfr / sbit*

```
unsigned char _data  *ucdp ; /* pointer to data in internal
ram */
unsigned char _code  *uccp ; /* pointer to data in R/O code
space */
unsigned char _idata *uccp;  /* pointer to upper 128 bytes
of ram */
```

All unqualified pointers are treated as 3-byte (4-byte for the ds390) *generic* pointers. These type of pointers can also to be explicitly declared.

```
unsigned char _generic *ucgp;
```

The highest order byte of the *generic* pointers contains the data space information. Assembler support routines are called whenever data is stored or retrieved usingrt routines generic

e)-285(will)-285(generater-285(more)-285(most)-285(ef)25(ficient)-285(code.)-415(Pointers)-285(declared)-285(usingr-285(a)]TJ 0 -11.955 Td[(mixtu

In the above example the variable *i* will be allocated in the external ram, *bvar* in bit addressable space and *j* in internal ram. When compiled with

3.7 Interrupt Service Routines

SDCC allows interrupt service routines to be coded in C, with some extended keywords.

In this case the address arithmetic a->b[i] will be computed only once; the equivalent code in C would be.

```
iTemp = a->b[i];
iTemp.c = 10;
iTemp.d = 11;
```

The compiler will try to keep these temporary variables in registers.

4.1.2 Dead-Code Elimination

```
int global;
void f () {
int i;
i = 1;  /* dead store */
global = 1; /* dead store */
global = 2;
return;
global = 3; /* unreachable */
}
```

will be changed to

```
int global; void f ()
{
global = 2;
return;
}
```

4.1.3 Copy-Propagation

```
int f() {
int i, j;
i = 10;
j = i;
return j;
}
```

will be changed to

```
int f() {
int i,j;
i = 10;
j = 10;
return 10;
}
```

Note: the dead stores created by this copy propagation will be eliminated by dead-code elimination.

4.1.4 Loop Optimizations

The more expensive multiplication is changed to a less expensive addition.

4.1.7 'switch' Statements

SDCC changes switch statements to jump tables when the following conditions are true.

The case labels are in numerical sequence, the labels need not be in order, and the starting number need not be one or zero.

```
switch(i) {
case 4:...
case 5:...
case 3:...
case 6:...
}
```

```
switch (i) {
            case 1:  ...
            case 2:  ...
            case 3:  ...
            case 4:  ...
}
```

4.1.9 Bit-rotation

A special case of the bit-shift operation is bit rotation, SDCC recognizes the following expression to be a left bit-rotation:

```
unsigned char i;
...
i = ((i << 1) | (i >> 7));
...
```

will generate the following code:

```
mov a,_i
rl a
mov _i,a
```

SDCC uses pattern matching on the parse tree to determine this operation.Variations of this case will also be recognized as bit-rotation, i.e.:

```
i = ((i >> 7) | (i C=CunsigJ1BensigJ1B;i1.526BT/F61 10.963 TfHighes50(be)-Order0(be)- 0 100
```

% heartily recommend this be the only way to get the highest order bit, (it is portable). Of course it will be recognized even if it is embedded in other expressions, e.g.:

```
xyz = gint + ((gint >> 15) & 1);
```

setjmp.h - contains de ntion for ANSI setjmp & longjmp routines. Note in this case setjmp & longjmp can be used between functions executing within the same register bank, if long jmp is executed from a function that is using a different register bank from the function issuing the setjmp function, the results may be unpredictable. The jump buffer requires 3 bytes of data (the stack pointer & a 16

stdlib.h - contains the following functions.

string.h - contains the following functions.

```
_asm_func:
push _bp
mov _bp,sp
mov r2,dpl
mov a,_bp
clr c
add a,#0xfd
mov r0,a
add a,#0xfc
mov r1,a
mov a,@r0
add a,r2
mov dpl,a
mov dph,#0x00
mov sp,_bp
pop _bp
ret
```

The compiling and linking procedure remains the same, however note the extra entry & exit linkage required for the assembler code, _bp is the stack frame pointer and is used to compute the offset into the stack for parameters and local variables.

4.5 External Stack

The external stack is located at the start of the external ram segment, and

```
foo()
{
...
s1 = s2 ; /* is invalid in SDCC although allowed in ANSI */
...
}
struct s foo1 (struct s parms) /* is invalid in SDCC although allowed
in ANSI */
{
struct s rets;
...
return rets;/* is invalid in SDCC although allowed in ANSI */
}
```

’long long’ (64 bit integers) not supported.

’double’ precisionoublfloomalongiot not supported.

SDCC uses the following formula to compute the complexity:

complexity = U050number of edges in control flow graphU051 - U050number of nodes in control flow graphU051 + 2;

Having said that the industry standard is 10, you should be aware that in some cases it be may unavoidable to have a complexity level of less than 10. For example if you have switch statement with more than 10 case labels, each case label adds one to the complexity level. The complexity level is by no means an absolute measure of the algorithmic complexity of the function, it does however provide a good starting point for which functions you might look at for further optimization.

5 TIPS

Here are a fe Td[(Ha379(guidelines)-378(that)-378(willHa379(help)-378(the)-378(compiler)-379(generate)-378(m

This phase determines the live-ranges; by live range I mean those iTemp variables de ned by the compiler that still survive after all the optimizations. Live range analysis is essential for register allocation, since these computation deter-

Phase ve is register allocation. There are two parts to this process.

7.3 Starting the Debugger

The debugger can be started using the following command line. (Assume the file you are debugging has the file name foo).

sdcdb foo

The debugger will look for the following files.

foo.c - the source file.

foo.cdb - the debugger symbol information file.

foo.ihx - the intel hex format object file.

7.4 Command Line Options.

–directory=<source file directory> this option can used to specify the directory search–direolook03250(o.944-298(the03298(directoch)-20429the03298(spfi(used)-250(for0329=<sou,for13(

```
;;
;; C-x SPC         sdcdb-break                  Set break for line with
point
```

10 Acknowledgments

Sandeep Dutta ¸sandeep.dutta@usa.net˚ - SDCC, the compiler, MCS11 code generator, Debugger, AVR port
Alan Baldwin ¸baldwin@shop-pdp.kent.edu˚ - Initial version of ASXXXX & ASLINE.

Index