SDCC for EdSim51

Pai H. Chou

Outline

- Download free C compiler SDCC
- Compile simple test program for EdSim51
- Data types
- Delay, I/O, logic, arithmetic operations

C Compilers for 8051

- SDCC: Small Device C Compiler
  http://sdcc.sourceforge.net
  - Open source, free, cross-platform => we'll use this
- Keil
  - free version has size limit; syntax difference
  - Used in EdSim51 examples
- IAR
  - limited-time (30-day) evaluation copy

Download/install SDCC (version 3.9.0 assumed)

- http://sourceforge.net/projects/sdcc/files/
- Unix: Extract the *.tar.gz
  - `tar xzf *.tar.gz`
  - set up the path to the binary
- Windows
  - Recommend: Cygwin for Unix-like environment
  - DOS version not recommended
- http://sdcc.sourceforge.net/doc/sdccman.pdf

SDCC

- "open source, retargetable, optimizing ANSI C compiler"
- Supported ISAs
  - Intel mds51 (by default),
    Zilog z80, Atmel AVR, TINI, Maxim ds390 & ds340, Motorola HC08, ...
- Experimental:
  - PIC (14-bit, 16-bit), ds400

Components of SDCC

- `sdcc` -- the C compiler
- `sdcpp` -- the C preprocessor
- `sdas8051` -- the 8051 assembler
- `sdld` -- the 8051 linker (link editor)
- `s51` -- the ucSim 8051 simulator
- `sdcdb` -- source debugger
- `sdar`, `sdranlib`, `sdnm`, `sdobjcopy` -- misc tools
- `packihx` -- packing Intel hex file

Data types in SDCC

Type	Width	Default	Range
bool	1 bit	unsigned	0, 1
char	1 byte	signed	-128 to 127
short	2 bytes		-32768 to 32767
int	2 bytes		-32768 to 32767
long	4 bytes		$-(2^{31})$ to $(2^{31})-1$
float	4 bytes		IEEE standard
pointer	1-4 bytes	n/a	0 to $(2^{bits}) - 1$

Unsupported Data Types

- Pointer to boolean
- Pass or return `struct` and `union` (but assignment is ok)
- Variable-length array
- `long long`,
- `long double`
- `double`

SDCC flags

- `sdcc -S` *`file`*`.c`
    - compile to assembly (`.asm`); don't assemble/link
- `sdcc -c` *`file`*`.c`
    - compile and assemble but don't link
    - creates relocatable object file (`.rel`)
    - good for separate compilations
- `-o` *`file`*`.hex`
    - name output file as *`file`*`.hex` instead of default name

Example of separate compilation and link

- Assume `delay.c` is used by several programs
  - `sdcc -c  delay.c`
    compile it once; makes file `delay.rel`
  - The `.rel` is relocatable object, unlinked
- Suppose `foo.c` wants to link with `delay.rel`
  - `sdcc -c foo.c`   // compile main
    `sdcc -o foo.hex foo.rel  delay.rel` // link
    - foo.hex is the final linked image

Example 1: test0.c

-
  ```
  #include <8051.h>
  void main(void)  {
      P1 = 0x24;
  }
  ```
- To compile (e.g., main.c), type
  - `sdcc main.c`
  - `packihx main.ihx > main.hex`

cleans up the hex file.
you can load it in EdSim51!

- creates .ihx .lnk .lst .map .mem .rel .rst .sym
- but.... will it work?

Output: .lst (or .asm)

```
__sdcc_program_startup:
        lcall   _main
        sjmp    .
```

```
_main:
        mov     _P1, #0x24
        ret
```

but.. SDCC's assembly syntax looks a little different, and it won't assemble if you paste into EdSim51

Output: .ihx file

- ihx = "Intel Hex" format
  - count, address, type, data, checksum

```
:03000000020006F5
:03005F0002000399
:0300030002006296
:04006200759024224F
:06003500E478FFF6D8FD9F
:200013007900E94400601B7A0090006A780175A000E493F2A308B8000205A0D9F4DAF27529
:02003300A0FF2C
:20003B007800E84400600A790175A000E4F309D8FC7800E84400600C7900900001E4F0A3C3
:04005B00D8FCD9FAFA
:0D000600758107120066E5826003020003A9
:04006600758200227D
:00000001FF
```

```
__interrupt_vect:
        ljmp    _sdcc_gsinit_startup
__sdcc_program_startup:
        ljmp    _main
```

```
_main:
        mov     _P1, #0x24
        ret
```

But.. this won't run on EdSim51, unless you make a few changes...

Startup Code

- Automatically linked in by linker for system initialization
  - however, assumes specific I/O features
  - To run in EdSim51, don't use any compiler-provided library
- Two alternative ways
  1. Simple: rename your main() as some other name, as long as it is the first
  2. More robust: define your own startup code, but keep main():
     - `void _sdcc_gsinit_startup(void) { main(); }`
     - `void _mcs51_genRAMCLEAR(void) { }`
     - `void _mcs51_genXINIT(void) { }`
     - `void _mcs51_genXRAMCLEAR(void) { }`
  - Note: just one _ in front of these function names! The compiler inserts another _ in front when generating assembly code

Compile & Link without default SDCC library

(1) change your main function to something other than main (e.g., Main)

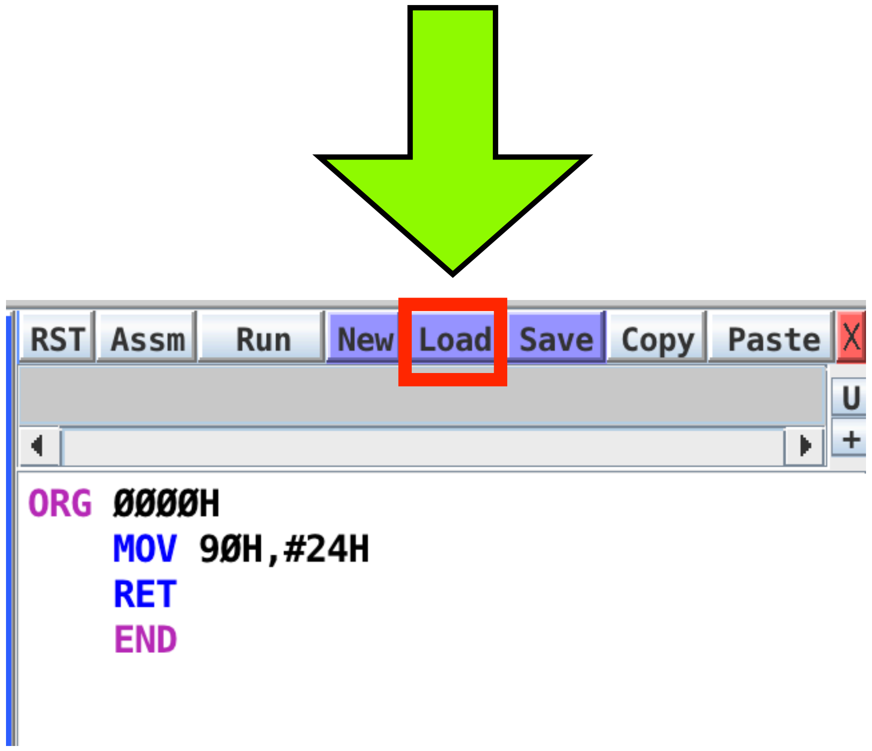
- alternatively, define startup code (see prev slide)

(2) compile and link as separate commands

- `sdcc -c test0.c` # compiles
- `sdcc test0.rel` # "links,"
- Load **`test0.hex`** into EdSim51
  - EdSim51 disassembles hex back to assembly (but without labels)

generated hex file

```
:0400000075902422B1
:00000001FF
```



```
ORG ØØØØH
    MOV 9ØH,#24H
    RET
    END
```

Modular (library) version of LED

```c
/* file LED7seg.c */
#include <8051.h>

char LED7seg(char num) {
	static __code char LEDdata[] = {
		0xC0, 0xF9, 0xA4, 0xB0, 0x99,
		0x92, 0x82, 0xF8, 0x80, 0x90
	};
	return LEDdata[num];
}
void DisplayLED(char num) {
	P1 = LED7seg(num);
}
```

```c
/* file: LED7seg.h */
#ifndef __LED7SEG_H__
#define __LED7SEG_H__

char LED7seg(char num);
void DisplayLED(char num);

#endif /* __LED7SEG_H__ */
```

```c
/* file: LEDtest0.c */
#include "LED7seg.h"
void Main(void) {
	char i;
	for (i = 0; i < 10; i++) {
		DisplayLED(i);
	}
}
```

Compile using the following commands

```
sdcc -c LEDtest0.c
sdcc -c LED7seg.c
sdcc -o LEDtest0.hex LEDtest0.rel LED7seg.rel
```

# note: `LEDtest0.hex` with Main() function must be linked first!

load `LEDtest0.hex` into EdSim51 and try! (LED7seg.rel is reusable!)

Keywords for Storage Classes

Storage class	where allocated
__data, __near	directly addressable internal RAM
__idata	indirectly addressable internam RAM
__bit	bit-addresable memory
__xdata, __far	external RAM
__pdata	paged: usually first 256 bytes in XData
__code	program memory
__sfr	special function register
__sbit	bit address in special function register

Example: Serial Echo

File: uartecho.c

```c
#include <8051.h>
void Main(void) {
   TMOD = 0x20;
   TH1 = -6;
   SCON = 0x50;
   TR1 = 1;
   while (1) {
      char c;
      while (!RI) { }
      c = SBUF;
      RI = 0;
      SBUF = c;
      while (!TI) { }
      TI = 0;
   }
}
```

- Compile with
  - `sdcc -c uartecho.c`
  - `sdcc -o uartecho.hex uartecho.rel`
- Remember to set
  - clock to 11.0592 MHz,
  - baud rate 4800
- Type into Tx window, see output from Rx window

Example: UART polling in C

Assembly "serialLED.asm"

```
           ORG 0H
           ;; initialize serial port
           MOV  TMOD, #20H    ;; to send
           MOV  TH1, #-6      ;; 4800 baud
           MOV  SCON, #50H    ;; 8-bit 1 stop REN
           SETB  TR1          ;; start timer 1
PollHere:  JNB    RI, PollHere ;; polling
           MOV  A, SBUF        ;; read serial port
           CLR   RI            ;; clear out receive flag
           ADD   A, #-48       ;; convert ASCII to binary
           LCALL  DisplayLED
           JMP     PollHere
Display:   MOV    DPTR, #LEDdata
           MOVC A, @A+DPTR  ;; A = LEDdata[A]
           MOV    P1, A         ;; light up LED seg
           RET                  ;; return from subroutine
LEDdata:   DB  0C0H, 0F9H, 0A4H, 0B0H, 99H, 92H, 82H
               0F8H, 80H, 90H
           END
```

C code "polluart.c"

```
#include <8051.h>
#include "LED7seg.h"
void Main(void) {
    TMOD = 0x20;
    TH1 = -6;
    SCON = 0x50;
    TR1 = 1;
    while (1) {
        char c;
        while (!RI) { }
        c = SBUF;
        RI = 0;
        DisplayLED(c - 48);
    }
}
```

Compile with =>
```
sdcc -c polluart.c
sdcc -o polluart.hex polluart.rel LED7seg.rel
```

load `polluart.hex` into EdSim51 to try

Example: Interrupt version of UART to LED

file: interLED.asm

```
•               ORG 0H
                JMP   Main ;; on startup, jump to main()
                ORG  23H  ;; this is the location for the ISR for serial port
                JMP   Serial_ISR
                ;; initialize serial port
Main:          LCALL  InitUart
                SETB   ES ;; enable interrupt for serial port
                SETB   EA ;; enable all interrupts
LoopHere: JMP   LoopHere      ;; infinite loop, could do useful work

•       Serial_ISR:             ;; make sure it's RI
                 JNB   TI, Check_RI
                 CLR   TI
Check_RI: JNB  RI, Serial_Done
                 MOV   A, SBUF          ;; read serial port
                 CLR   RI               ;; clear out receive flag
                 ADD   A, #-48          ;; convert ASCII to binary
                 LCALL  Display         ;; update the display
Serial_Done:   RETI                     ;; return from ISR
```

file: intrLED.c

```
#include <8051.h>
#include "LED7seg.h"
char RxData; // the received data
void InitUart(void) {
        TMOD = 0x20;
        TH1 = -6;
        SCON = 0x50;
        TR1 = 1;
}
void main(void) {
        InitUart();
        EA = 1; // enable all interrupts
        ES = 1; // enable serial interrupt
        while (1) { }
}

void Serial_ISR(void) __interrupt(4) {
        if (TI) {
                TI = 0;
        }
}
```

```
void _sdcc_gsinit_startup(void) {
        __asm
                mov sp, #0x57
        __endasm;
        main();
}
void _mcs51_genRAMCLEAR(void) {}
void _mcs51_genXINIT(void) {}
void _mcs51_genXRAMCLEAR(void)
{}
```

Compile with

```
sdcc -c intrLED.c
sdcc -o intrLED.hex intrLED.rel LED7seg.rel
```

how the code works

```
void _sdcc_gsinit_startup(void) {
        __asm
                mov sp, #0x57
        __endasm;
        main();
}
void _mcs51_genRAMCLEAR(void) {}
void _mcs51_genXINIT(void) {}
void _mcs51_genXRAMCLEAR(void) {}
```

- on startup,
  - reposition stack pointer (SP)
  - jump to main()
  - Necessary to jump out of reset handler to actual main(), instead of relying on Main() to be located as the reset handler!
- Other required routines
  - `_mcs51_genRAMClear(); _mcs51_genXINIT(); _mcs51_genXRAMCLEAR();` => by declaring them, we override the library version!

Syntax of ISR in SDCC

```
void Serial_ISR(void) __interrupt(4) {
        if (TI) {
                TI = 0;
        }
        if (RI) {
                RxData = SBUF;
                RI = 0;
                DisplayLED(RxData-48);
        }
}
```

compare asm with hand-crafted code..

```
Serial_ISR:              ;; make sure it's RI
            JNB   TI, Check_RI
            CLR   TI
Check_RI:   JNB   RI, Serial_Done
            MOV   A, SBUF          ;; read serial port
            CLR   RI               ;; clear out receive flag
            ADD   A, #-48          ;; convert ASCII to binary
            LCALL DisplayLED       ;; update the display
Serial_Done:  RETI                 ;; return from ISR
```

- __interrupt(4)
  - declare as ISR for interrupt#4 (UART)
- code using RETI
  - instead of RET
- Issue: calling DisplayLED() from ISR...

Issues with ISR

- Register saving and restoring
  - R0..R7 needs to be saved (if used)
  - PSW, ACC, B register, ... non-I/O ones should be saved
- Duration of ISR
  - should minimize amount of work spent in ISR
  - ISR should do just enough transfer, leave longer task to be done in user code
  - Therefore, calling DisplayLED() from ISR might not be a good idea!

Look at SDCC code for ISR: (1/3) preamble

- Look in the `intrLED.lst` file
- 388 _Serial_ISR:
- 389 push bits
- 390 push acc
- 391 push b
- 392 push dpl
- 393 push dph
- 394 push (0+7)
- 395 push (0+6)
- 396 push (0+5)
- 397 push (0+4)
- 398 push (0+3)
- 399 push (0+2)
- 400 push (0+1)
- 401 push (0+0)
- 402 push psw

Look at SDCC code for ISR: (2/3) body code

```
	mov	psw,#0x00
;	if (TI) {
;		TI = 0;
	jbc	_TI,00113$
	sjmp	00102$
00113$:
00102$:
;	if (RI) {
	jnb	_RI,00105$
;		RxData = SBUF;
	mov	_RxData,_SBUF
;	RI = 0;
	clr	_RI
;DisplayLED(RxData-48);
	mov	a,_RxData
	add	a,#0xD0
	mov	dpl,a
	lcall	_DisplayLED
00105$:
```

Look at SDCC code for ISR: (3/3) post amble

- 421 00105$:
- 422 pop psw
- 423 pop (0+0)
- 424 pop (0+1)
- 425 pop (0+2)
- 426 pop (0+3)
- 427 pop (0+4)
- 428 pop (0+5)
- 429 pop (0+6)
- 430 pop (0+7)
- 431 pop dph
- 432 pop dpl
- 433 pop b
- 434 pop acc
- 435 pop bits
- 436 reti

which registers to save?

- save only those that the ISR will affect!
  - Likely affected: PSW, ACC, maybe DPTR?
  - Save only those R0..R7 if actually used
- No need to save extra ones
  - save more => more stress on stack! (likely to overflow)