Compilers & Tools

Outline

- Tools & Work Flow
- Language & Storage Class Extensions
- Calling convention
- inlined assembly
- Library Support

8051 programming in C

- Download free C compiler SDCC
- Compile simple test program
- 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
- Keil
  - free version has size limit; syntax difference
  - Used in EdSim51 examples
- IAR
  - limited-time (30-day) evaluation copy

Download/install SDCC (version 3 assumed)

- http://sourceforge.net/projects/sdcc/files/
- Unix: Extract the *.tar.gz
  - tar xzf *.tar.gz
  - set up the path to the binary
- Windows
  - run the *-setup.exe
    Open the DOS prompt to run sdcc
  - Recommend: Cygwin for Unix-like environment
- 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
- 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 $(2^{\text{bits}}) - 1$

Unsupported Data Types

- Pointer to boolean
- Pass or return struct and union
- 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.ihx
  - name output file as file.ihx 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 delay.rel
  - The .rel is relocatable object, unlinked
- Suppose foo.c wants to use it
  - sdcc -c foo.c   // compile main
    sdcc -o foo.ihx foo.rel delay.rel // link

Example 1: main.c

-
  ```
  #include <8051.h>
  void main(void)  {
      P1 = 0x12;
  }
  ```
- 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

Output: .lst (or .asm)

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

```
_main:
                mov         _P1, #0x12
                ret
```

Paste the .ihx (Intel Hex) file into EdSim to run!
if you want, rename it with .hex suffix

Startup Code

- There is a lot of code between

```
    ORG    000H
    LJMP   0006H  ;;
    LJMP   0062H  ;;
    MOV    81H, #07H
    LCALL  0065H   ;; calling main()
```

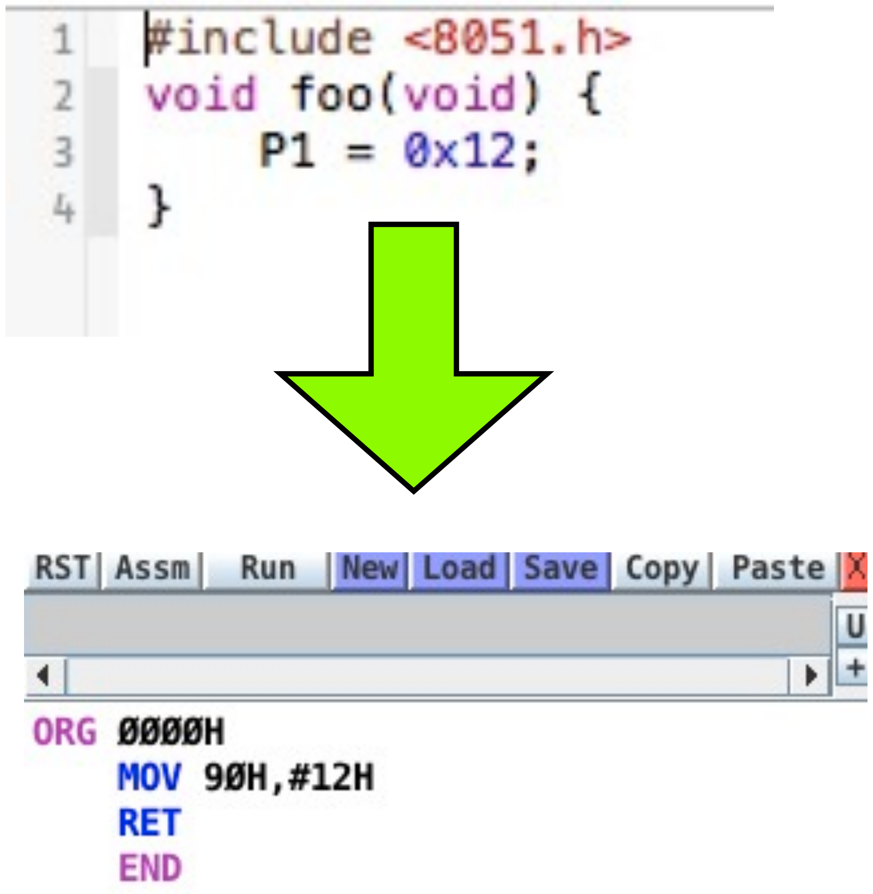
  and

```
0065H:   MOV 90H, #12H;;main:  P1=0x12
```

- Why? because the compiler automatically links in system-init code!

What if you don't want startup code?

- (1) name your function something other than main
- (2) compile and link as separate commands
- sdcc -c foo.c   # compiles
- sdcc foo.rel   # "links,"



```
#include <8051.h>
void foo(void) {
        P1 = 0x12;
}
```

```
ORG 0000H
    MOV 90H,#12H
    RET
    END
```

Example 2: signed char

```
#include <8051.h>
void Main(void)  {
     char mynum[]= {+1,-1,+2,-2,+3,-3,+4,-4};
     unsigned char z;
     for (z = 0; z < 8; z++) {
          P1 = mynum[z];
     }
}
```

Example 2 variation: local vs. global const

```
#include <8051.h>
const char mynum[]= {+1,-1,+2,-2,+3,-3,+4,-4};
void Main(void) __naked {
    ~~char mynum[]= {+1,-1,+2,-2,+3,-3,+4,-4};~~
    unsigned char z;
    for (z = 0; z < 8; z++) {
        P1 = mynum[z];
    }
}
```

how does that change the generated code?

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 declaration with storage class

- __data unsigned char d;
- __xdata unsigned char x;
- __idata unsigned char i;
- __pdata unsigned char p;
- __code unsigned char t[] = {'a', 'b', 'c'};
- __bit mybit; // implies boolean

MCU-specific features

- 8051-specific Declarations
  - **__sfr __at** (*address*) *name;*
    __sbit __at (*address*) *name;*
  - e.g., __sfr __at (0x80) P0;
    __sbit __at (0x81) P0_1; // P0.1
- Possible to concatenate two to make 16 bits!
  - __sfr16 __at (0x8382) DPTR;
    // DPH = 0x83, DPL = 0x82

example 3: sbit, int

```
#include <8051.h>
#define MYBIT P1_0  // for port P1.0
void Main(void)  {
    unsigned int z;
    for (z = 0; z < 50000; z++) {
        MYBIT = 0; MYBIT = 1;
    }
}
```

Issues with example 3

- Keil C uses syntax **sbit** MYBIT = P1^0;
  - but P1^0 is an xor expression!
    in SDCC, use P1_0
  - it's like allocate a bit, initialize to P1_0
    probably not what is intended!
- To declare bit in SDCC syntax,
  __sbit __at 0x90 MYBIT;
  - Define MYBIT to be bit at address 90H

example 4: comparison

```
#include <8051.h>
void Main(void)  {
        unsigned char mybyte;
        P0 = 0xFF;
        while (1) {
              if (P0 < 100) {
                    mybyte = P1;
              } else {
                    mybyte = P2;
              }
        }
}
```

Q: What if you use P0 directly?
if (P0 < 100)
instead of copying into mybyte first?
is P0 treated as signed or unsigned?

example 5: toggle bit

```
__sbit __at 0xA4  mybit;
void Main(void) {
        mybit = 1;
        while (1) {
                mybit = ! mybit;
        }
}
```

- don't use ~ operator to complement; use !
- ~ will promote bit to **int** (or **char**)

Declaring SFRs

```
__sfr __at (0x80) P0;
__sfr __at (0x90) P1;
__sfr __at (0xA0) P2;
__sbit __at (0x95) P1_5;
```

- If you declare **__sfr** this way, you don't need to #include <8051.h>

ex.6: bit vs. sbit

```
__sbit __at 0x90 inbit; // P1.0
__sbit __at 0xA7 outbit; // P2.7
__bit membit;
void Main(void){
    while (1) {
        membit = inbit;
        outbit = membit;
    }
}
```

- sbit is specified with a bit address
- bit is automatically allocated by the compiler to any bit addressable memory.
- You could specify bit address if you want

ex.7: bitwise operators

```
#include <8051.h>
void Main(void) {
    P0 = 0x35 & 0xF;
    P1 = 0x04 | 0x68;
    P2 = 0x54 ^ 0x78;
    P0 = ~0x55;
    P1 = 0x9A >> 3;
    P2 = 0x77 >> 4;
    P0 = 0x06 << 4;
}
```

- Guess what: Compiler performs constant folding!
- All of these expressions get evaluated at compile time
- assembly: just MOV of constants to ports

Toggling bits

- First way: the ~ operator
  - P0 = ~P0;
- Second way: xor with 1's (^ operator)
  - P0 = P0 ^ 0xFF;
  - alternatively, P0 ^= 0xFF;

ex.8: inverting a bit

```
#include <8051.h>
#define inbit P1_0
#define outbit P2_7
__bit membit;
void Main(void) {
    while (1) {
        membit = inbit;
        outbit = ~membit;
    }
}
```

- You will get a compiler warning about ~
  - You can get unexpected result due to promotion to int
- use ! instead of ~ for inverting a single bit

ex.9: switch statement

```
#include <8051.h>
void Main(void) {
    switch (P1 & 0x3) {
    case 0: P0='0'; break;
    case 1: P0='1'; break;
    case 2: P0='2'; break;
    case 3: P0='3'; break;
    }
}
```

- No need to use separate variable z to store P1; should be able to use it in an expression
- Look at the .asm for the switch statement: does it use a jump table or cascaded conditionals?

Memory spaces (review)

- On-chip RAM: MOV @R(1-byte pointer)
  - 0-7FH: reg, bit-addressable, scratchpad
  - 8052 has indir. addressable only 80-FFH
- Code space: MOVC @DPTR (2-byte ptr)
- Ext. RAM: MOVX @DPTR (2-byte ptr)

Keywords for data models

- **__data**: (by default) direct-addressable internal RAM (0-7FH)
  e.g., **__data char** d[] = "Hello";
  can also use **__near**, same as **__data**
  e.g., **__near char** d[] = "Hello";

- **_idata**: (8052's) indir. addr. RAM (80-FFH)
  e.g., **__idata char** d[] = "Hello";

- The double-underscore **__data**, **__idata** are for ANSI compliance, started in v3.0

Keywords for data models (cont'd)

- **xdata**: external RAM
  e.g., **__xdata char** d[] = "Hello";
  __xdata and **__far** mean the same
  e.g., **__far char** d[] = "Hello";

- **code**: code ROM
  e.g., **__code char** d[] = "Hello";

- There is also **__pdata**, which is "paged" external data (will revisit later)

ex.10a: compute checksum

```
#include <8051.h>
void Main(void) {
  unsigned char d[] = {0x25,0x62,0x3F,0x52};
  unsigned char sum=0;
  unsigned char x;
  for (x=0; x<sizeof(d); x++) {
        P2 = d[x];
        P1 = sum += d[x];
  }
  P1 = (~sum) + 1
}
```

- Another way of writing the same code
- use function calls to avoid constant folding

Example ex.10b: verify checksum

```
#include <8051.h>
void Main(void) {
  unsigned char d[] = {0x25,0x62,0x3F,0x52,0xE8};
  unsigned char sum=0;
  unsigned char x;
  for (x=0; x<sizeof(d); x++) {
      sum += d[x];
  }
  P0 = sum ? 'B' : 'G';
}
```

ex.11: Convert byte to decimal digits

```
#include <8051.h>
void convertToDecimal(unsigned char b) {
  unsigned char quo;
  quo = b / 10;
  P2 = quo / 10;  /*  hundreds */
  P1 = quo % 10;  /* tens */
  P0 = b % 10;  /*  ones */
}
```

Other language features

- Binary constants
  - (compile w/ --std-sdccxx)
  - e.g., 0b01100010 (=0x62)
- Volatile
  - volatile __xdata __at (0x8000) xsfr8k;

Inlined assembly

-
  ```
  {
  __asm
      mov r2, dpl
      mov a, #2
      mov r3, a
      ...
  __endasm;
  ...}
  ```

Library support

- #include <stdio.h>
    - getchar(), putchar() -- works with serial port
    - printf() -- does not support float
- #include <malloc.h>
- #include <math.h>