C Language Programming

for the 8051

Overview

- C for microcontrollers
  - Review of C basics
  - Compilation flow for SiLabs IDE
  - C extensions
  - In-line assembly
  - Interfacing with C
- Examples
- Arrays and Pointers
- I/O Circuitry
- Functions and Header Files
- Multitasking and multithreading

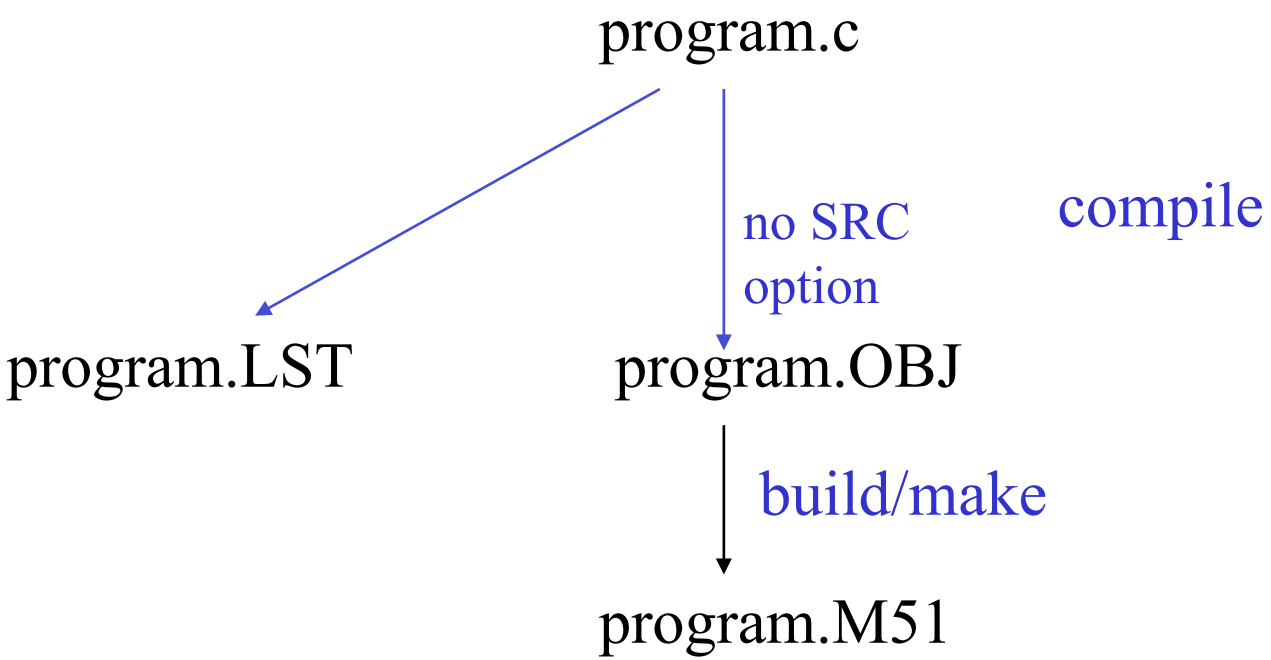
C for Microcontrollers

- Of higher level languages, C is the closest to assembly languages
  - bit manipulation instructions
  - pointers (indirect addressing)
- Most microcontrollers have available C compilers
- Writing in C simplifies code development for large projects.

Available C Compilers

- Kiel – integrated with the IDE we have been using for labs.
- Reads51 – available on web site (http://www.rigelcorp.com/reads51.htm)
- Freeware: SDCC - Small Device C Compiler (http://sdcc.sourceforge.net/)
- Other freeware versions …

Compilation Process (Keil)



Modular Programming

- Like most high level languages, C is a modular programming language (but NOT an object oriented language)
- Each task can be encapsulated as a function.
- Entire program is encapsulated in “main” function.

Basic C Program Structure

1. Compiler directives and include files
2. Declarations of global variables and constants
3. Declaration of functions
4. Main function
5. Sub-functions
6. Interrupt service routines

Example: blinky.c

Back to C Basics

- All C programs consists of:
  - Variables
  - Functions (one must be “main”)
    - Statements

- To define the SFRs as variables:

```
#include <c8051F020.h>
```

Variables

- All variables must be declared at top of program, before the first statement.
- Declaration includes *type* and list of variables.

```
Example:    void main (void) {
                int var, tmp;
```

must go HERE!

- Types:
  - **int** (16-bits in our compiler)
  - **char** (8-bits)
  - **short** (16-bits)
  - **long** (32-bits)
  - **sbit** (1-bit)
  - others that we will discuss later

not standard C – an 8051 extension

Variables

- The following variable types can be signed or unsigned:

signed char (8 bits) –128 to +127
signed short (16 bits) –32768 to +32767
signed int (16 bits) –32768 to +32767
signed long (32 bits) –2147483648 to +2147483648

unsigned char (8 bits) 0 to + 255
unsigned short (16 bits) 0 to + 65535
unsigned int (16 bits) 0 to + 65535
unsigned long (32 bits) 0 to + 4294967295

NOTE: Default is signed – it is best to specify.

Statements

- Assignment statement:

variable = *constant* or *expression* or *variable*

examples: upper = 60;

I = I + 5;

J = I;

Operators

- Arithmetic: +, -, *, /
- Relational comparisons: >, >=, <, <=
- Equality comparisons: ==, !=
- Logical operators: && (and), || (or)
- Increment and decrement: ++, --
- Example:

```
if (x != y) && (c == b)
{
    a=c + d*b;
    a++;
}
```

Example – Adder program (add 2 16-bit numbers)

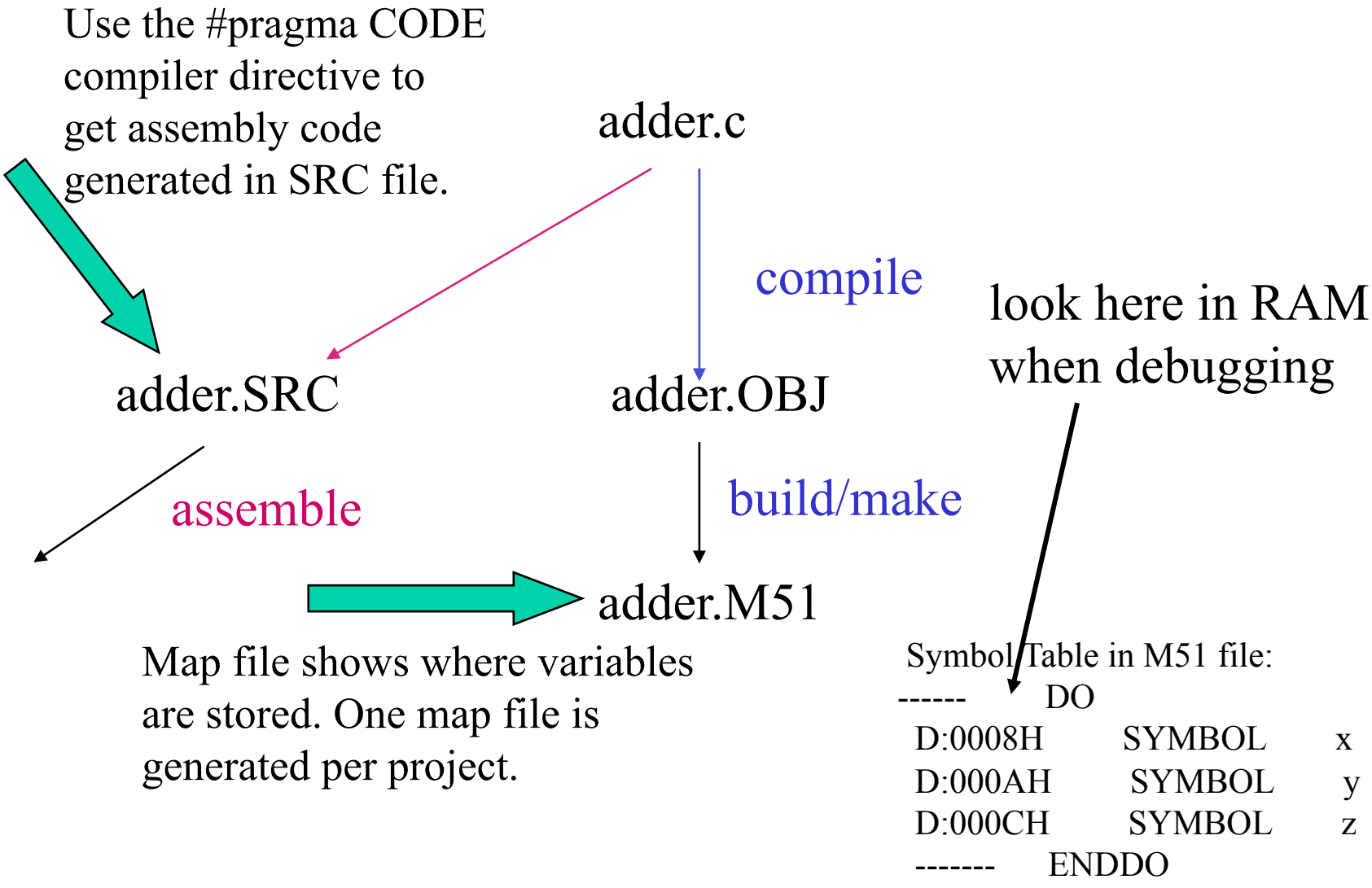
```
$INCLUDE (C8051F020.inc)
XL equ 0x78
XH equ 0x79
YL equ 0x7A
YH equ 0x7B
     cseg at 0
     ljmp Main
cseg at 100h
; Disable watchdog timer
Main:       mov 0xFF, #0DEh
            mov 0xFF, #0ADh
            mov a, XL
            add a, YL
            mov XL, a
            addc a, YH
            mov XH, a
            nop
end
```

The assembly version

```
#include <c8051f020.h>
void main (void) {
int x, y, z;   //16-bit variables
   // disable watchdog timer
   WDTCN = 0xde;
   WDTCN = 0xad;
   z = x + y;
}
```

The C version

Compilation Process (Keil)



adder.SRC

```
          x?040:    DS   2
          y?041:    DS   2
          z?042:    DS   2
main:
                            ; SOURCE LINE # 12
; int x, y, z;
;         WDTCN = 0xde;               // disable watchdog timer
                            ; SOURCE LINE # 14
          MOV       WDTCN,#0DEH
;         WDTCN = 0xad;
                            ; SOURCE LINE # 15
          MOV       WDTCN,#0ADH
;         z = x + y;
                            ; SOURCE LINE # 17
          MOV       A,x?040+01H
          ADD       A,y?041+01H
          MOV       z?042+01H,A
          MOV       A,x?040
          ADDC      A,y?041
          MOV       z?042,A
;         }                           ; SOURCE LINE # 18
          RET
; END OF main
          END
```

Bitwise Logic Instructions

		Examples:
• AND	&	n = n & 0xF0;
• OR	|	
• XOR	^	
• left shift	<<	n = n & (0xFF << 4)
• right shift	>>	
• 1's complement	~	n = n & ~(0xFF >> 4)

Example – Logic in Assembly and C

```
Main:
    mov WDTCN, #0DEh
    mov WDTCN, #0ADh
    xrl a, #0xF0     ; invert bits 7-4
    orl a, #0x0C     ; set bits 3-2
    anl a, #0xFC    ; reset bits 1-0
    mov P0, a      ; send to port0
```

```
void main (void) {
char x;
WDTCN = 0xDE;
WDTCN = 0xAD;
x = x ^ 0xF0;
x = x | 0x0C;
x = x & 0xFC;
P0 = x;
}
```

Loop Statements - While

- While loop:

  while (condition) { statements }

  while condition is true, execute statements

  if there is only one statement, we can lose the {}

  Example: while (1) ; // loop forever

Loop Statements - For

- For statement:

for (initialization; condition; increment) {statements}

initialization done before statement is executed

condition is tested, if true, execute statements
do *increment* step and go back and test condition again

repeat last two steps until condition is not true

Example: for loop

```
for (n = 0; n<1000; n++)
```

n++ means n = n + 1

Be careful with signed integers!

```
for (i=0; i < 33000; i++) LED = ~LED;
```

Why is this an infinite loop?

Loops: do - while

```
do
   statements
while (expression);
```

Test made at the bottom of the loop

Decision – if statement

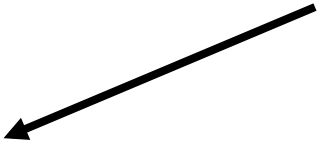
```
if (condition1)
    {statements1}
  else if (condition2)
    {statements2}
  …
  else
    {statementsn}
```

Decision – switch statement

```
switch (expression) {
    case const-expr: statements
    case const-expr: statements
    default: statements
}
```

Example: switch

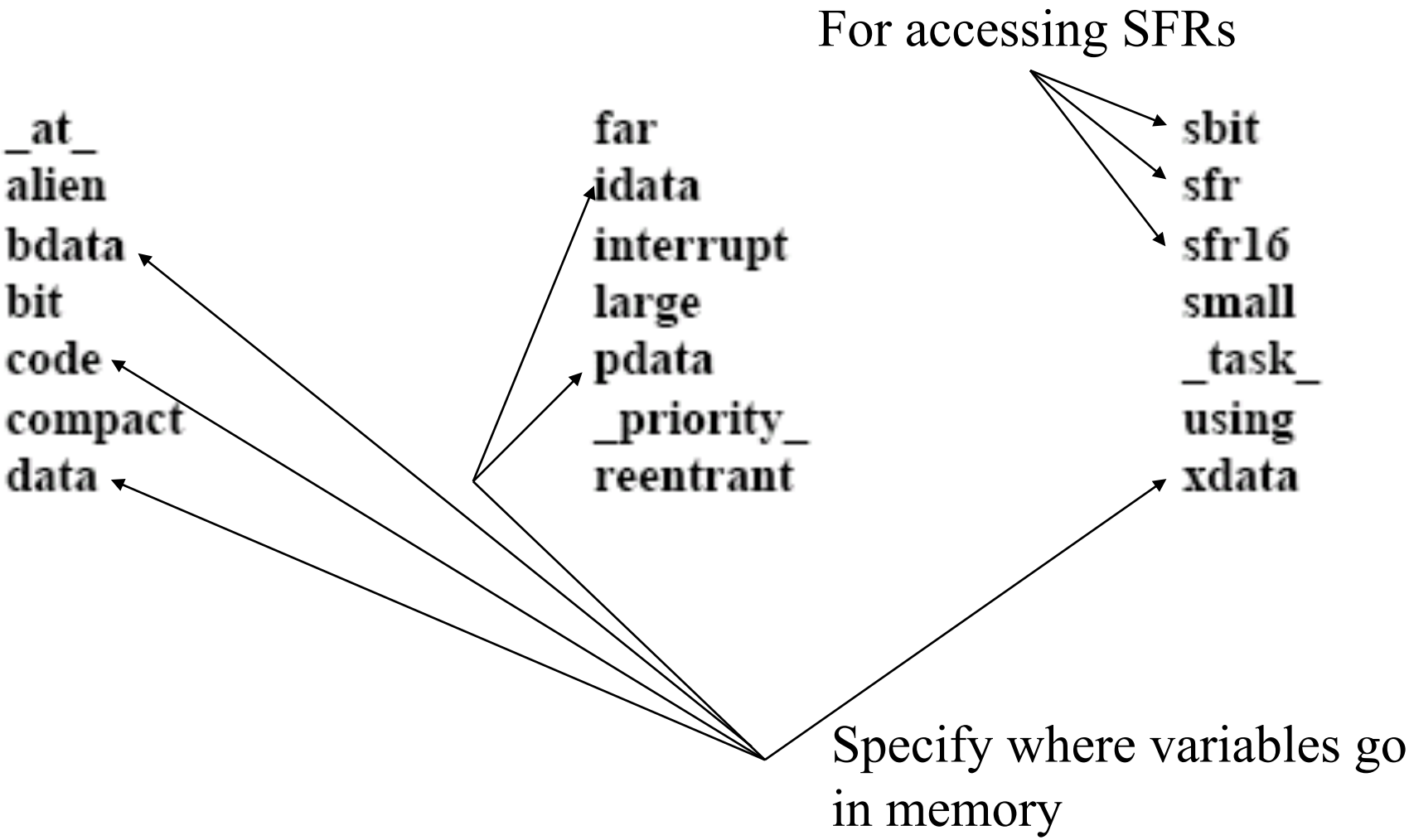
```
switch (unibble) {
    case 0x00 : return (0xC0);
    case 0x01 : return (0xF9);
    case 0x02 : return (0xA4);
    case 0x03 : return (0xC0);
        default : return (0xFF);
}
```



Need a statement like “return” or “break” or execution falls through to the next case (unlike VHDL)

Revisit Toggle and Blink5

C Extensions: Additional Keywords

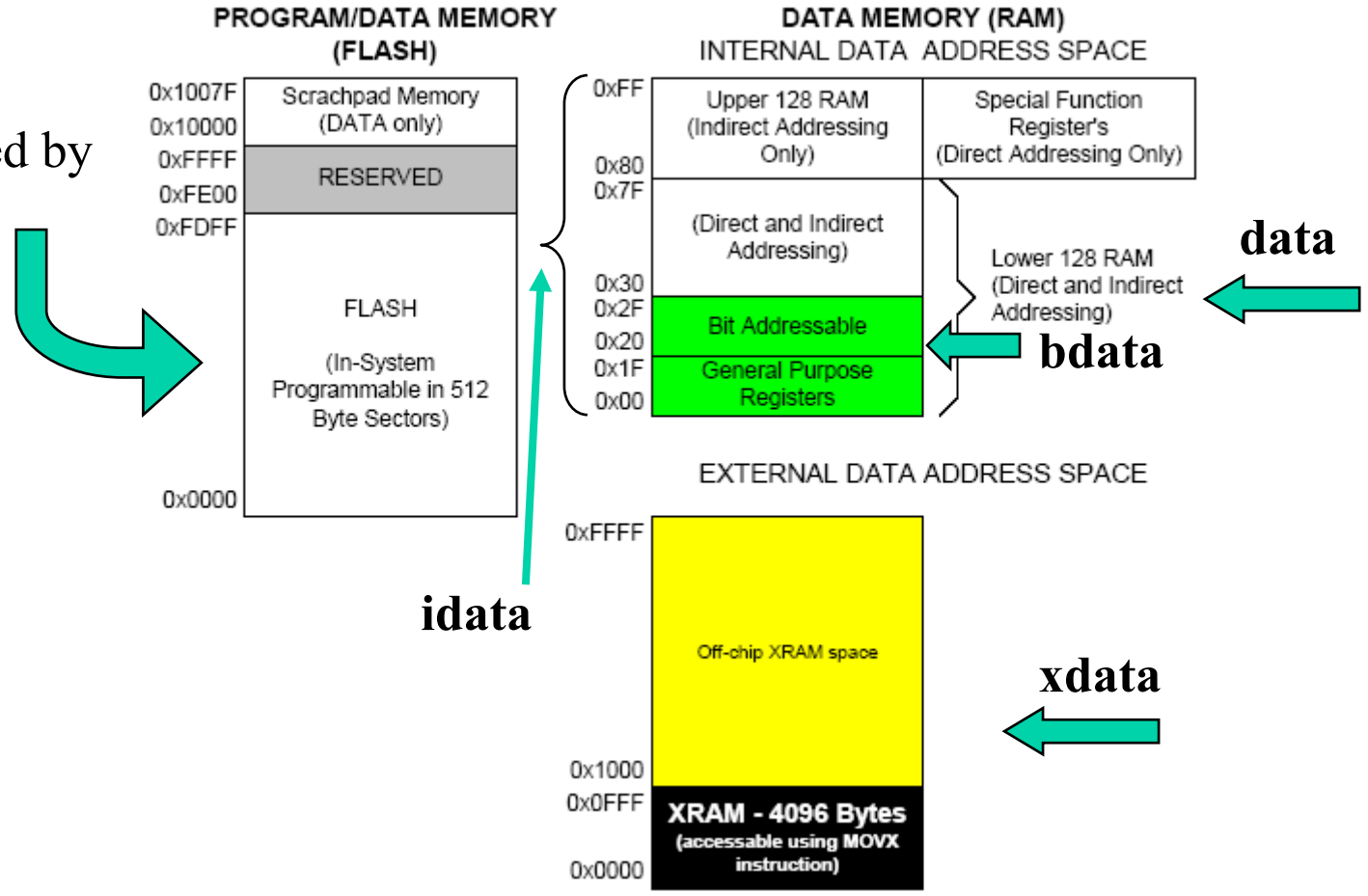


Accessing Specific Memory

Memory Type	Description
code	Program memory (64 KBytes); accessed by opcode MOVC @A+DPTR.
data	Directly addressable internal data memory; fastest access to variables (128 bytes).
idata	Indirectly addressable internal data memory; accessed across the full internal address space (256 bytes).
bdata	Bit-addressable internal data memory; supports mixed bit and byte access (16 bytes).
xdata	External data memory (64 KBytes); accessed by opcode MOVX @DPTR.
far	Extended RAM and ROM memory spaces (up to 16MB); accessed by user defined routines or specific chip extensions (Philips 80C51MX, Dallas 390).
pdata	Paged (256 bytes) external data memory; accessed by opcode MOVX @R*n*.

C Access to 8051 Memory

code: program memory accessed by movc @a + dptr



C Extensions for 8051 (Cygnal)

- New data types:

	Example:
bit	bit new_flag; //stored in 20-2F
sbit	sbit LED = P1^6;
sfr	sfr SP = 0x81; //stack pointer
sfr16	sfr16 DP = 0x82; // data pointer

$INCLUDE (c8051F020.h)

C Data Types With Extensions

Data Types	Bits	Bytes	Value Range
bit †	1		0 to 1
signed char	8	1	-128 to +127
unsigned char	8	1	0 to 255
enum	8 / 16	1 or 2	-128 to +127 or –32768 to +32767
signed short	16	2	-32768 to +32767
unsigned short	16	2	0 to 65535
signed int	16	2	-32768 to +32767
unsigned int	16	2	0 to 65535
signed long	32	4	-2147483648 to +2147483647
unsigned long	32	4	0 to 4294967295
float	32	4	±1.175494E-38 to ±3.402823E+38
sbit †	1		0 or 1
sfr †	8	1	0 to 255
sfr16 †	16	2	0 to 65535

Declaring Variables in Memory

```
char data temp;
char idata varx;
int xdata array[100];
char code text[] = “Enter data”;
```

Example: Accessing External Memory

- Program defines two 256 element arrays in external memory
- First array is filled with values that increase by 2 each location.
- First array is copied to second array.
- Similar to block move exercise done in assembly.
- xdata_move.c

Interrupts – Original 8051

Interrupt Number	Interrupt Description	Address
0	EXTERNAL INT 0	0003h
1	TIMER/COUNTER 0	000Bh
2	EXTERNAL INT 1	0013h
3	TIMER/COUNTER 1	001Bh
4	SERIAL PORT	0023h

Specify register bank 2

```
void timer0 (void) interrupt 1 using 2 {
        if (++interruptcnt == 4000) {          /* count to 4000 */
        second++;                              /* second counter */
        interruptcnt = 0;                      /* clear int counter */
        }
}
```

Other Interrupt Numbers

Interrupt Number	Address
0	0003h
1	000Bh
2	0013h
3	001Bh
4	0023h
5	002Bh
6	0033h
7	003Bh
8	0043h
9	004Bh
10	0053h
11	005Bh
12	0063h
13	006Bh
14	0073h
15	007Bh

Interrupt Number	Address
16	0083h
17	008Bh
18	0093h
19	009Bh
20	00A3h
21	00ABh
22	00B3h
23	00BBh
24	00C3h
25	00CBh
26	00D3h
27	00DBh
28	00E3h
29	00EBh
30	00F3h
31	00FBh

Interrupt number is same as “Priority Order” in datasheet