**Hitachi America, Ltd.**
**Application Engineering**
**TechNotes**

October, 1991

Paul Yiu

# Direct Memory Addressing with C Pointers

## Direct Addressing

In Embedded Programming, the C source code often needs to access memory addresses directly to drive the hardware peripherals, such as I/O ports, timers, registers, etc.

The best way to demonstrate direct addressing is by example. If we have a timer, called tmr0, at addressing 0xffc8, and we want to change its value. This is the simplest way:

1 Define tmr0 as a constant:

In C source code:

***unsigned int const tmr0 = 0xFFC8;***

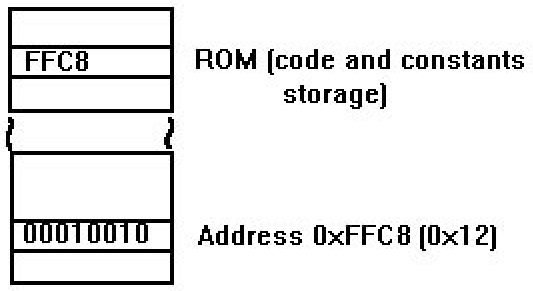
a) It's best to define addresses as unsigned numbers, so the compiler won't mistake 0xffc8 as a negative number.
b) The compiler puts our variable, tmr0, in ROM because it's declared as a constant, thus freeing up more RAM.
c) In C, hex numbers are preceded by 0x(zero x).
d) Pointers are extremely tricky, and they take up extra memory space, so we define tmr0 as just an integer. We can cast this number to be a pointer later.

2 Cast this integer as a pointer to a memory address:

In C source code:

****(unsigned char *)tmr0 = 0x12;***

a) ***(unsigned char *)*** casts tmr0 to be a pointer to an unsigned character. Now, (unsigned char *)tmr0 refers to address hex FFC8. Adding a '*' in front of it makes the expression content of address 0xFFC8.



b) If we want to put an integer in address FFC8 and FFC9, we can just change the casting to (unsigned int *), then if we say ***(unsigned int *)tmr0=0x1234;***, we will have 0x12 in FFC8, 0x34 in FFC9.
c) The benefit of defining address as integers instead of pointers is we don't need to define this pointer as a character or an integer until we want to put numbers in these addresses. These integers, casted as pointers, are much more flexible.

**Pointers to Strings**

In C, the simplest way to define a string is to define a pointer that points to that string.

In C source code:

```
unsigned char *sinatra=”Fly me to the moon.”;
```

Compiled:

```
.EXPORT _sinatra
_sinatra    .DATA.W       S0
.SECTION          strings,TEXT,ALIGN=2
S0  .SDATA          “Fly me to the moon”<0>
```

a) The compiler will put this string in the strings section. The pointer, “sinatra” refers to the address of the first character, “F.”
b) *sinatra refers to the character “F.”

Say we have defined another pointer, called ptr.

In C source code:

```
unsigned char *ptr;
ptr=sinatra;
```

Compiled:

```
.IMPORT _ptr
.comm _ptr,H’2
mov.w       @_sinatra,r0
mov.w       r0,@_ptr
```

a) Here, we first define a pointer, called ptr, not pointed to anything yet. Secondly, we tell the compiler to let “ptr” point to whatever “sinatra” is pointing to. *ptr now is character “***F.***” Since the string is stored in memory in order, if we increment ptr, we get the next character.

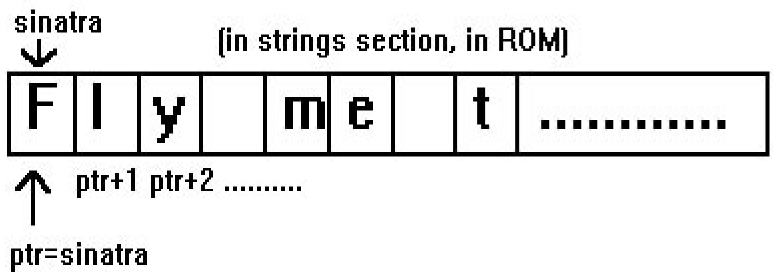
In C source code:

```
ptr++;
```

Compiled:

```
adds          #1,r0
mov.w       r0,@_ptr
```

a) *ptr now is character “ ***l***.”
b) ptr corresponds to the address of the letter “***l***.”



The information in this document has been carefully checked; however, the contents of this document may be changed and modified without notice. Hitachi America, Ltd. shall assume no responsibility for inaccuracies, or any problem involving a patent infringement caused when applying the descriptions in this document. This material is protected by copyright laws. Hitachi America, Ltd. reserves all rights.