

Keil C51 μ Vision4 的使用

Keil C51 μ Vision4 是美国 Keil Software 公司出品的 51 系列兼容单芯片 C 语言整合发展环境(Integrated Development Environment, IDE)，与汇编语言相比，C 语言在功能上、架构性、可读性、可维护性上有明显的优势，因而易学易用。用过汇编语言后再使用 C 语言来开发，会更加体会深刻。目前最新的版本是 C51 Version 9.00 Release，也就是所谓的 μ Vision4。

Keil C51 软件提供丰富的函数库和功能强大的除错工具，及全窗口界面。另外重要的一点，只要看一下编译后生成的汇编语言代码，就能体会到 Keil C51 生成的目标代码效率非常之高，多数语句生成的汇编语言代码很紧凑，容易理解。在开发大型软件时更能体现高级语言的便利与优势。

Keil C51 μ Vision4 较之前的版本，新增了下面几个功能：

- 多重的监控和弹性的窗口管理系统。

- 系统查看器(System Viewer) - 显示设备周边缓存器的讯息。除错恢复检视

- (Debug Restore Views) - 建立和储存多重除错窗口设计。多项目工作区(Multi-

- Project Workspace) - 与许多项目简化工作。源码和解组译连结(Source and

- Disassembly Linking) - 解组译窗口和源码窗口完全同步使程序除错和光标导航较容易。

- 内存窗口固定(Memory Window Freeze) - 储存目前内存窗口检视允许容易在不同的点及时比较。

- 设备模拟 - 更新支持很多新设备(例如 Infineon XC88x, SiLABS C8051Fxx, Atmel SAM7/9 和从 Luminary, NXP, and Toshiba 来的 Cortex M3 MCUs)。支持硬件

- 除错转接器(Support for Hardware debug adapters) - 包括 ADI。

- miDAS-Link, Atmel SAM-ICE, Infineon DAS, 和 ST-Link。新资料和指令追踪(New Data and instruction trace) - 对 ARM 和 Cortex MCUs。

- 基于 XML 的项目文件(XML based Project files) - 建立、检视和修改项目如同容易可读的 XML 本文档一样。

- 串列的窗口 - 扩充到提供一个基本的 100-VT 终端机，ASCII 模式，混合模式，和十六进制模式检视。

- 拖放档案开启(Drag & Drop File Opening) - 档案拖进 μ Vision4 项目空间自动会被开启。

- 监控点和逻辑分析仪(Watchpoints and Logic Analyzer) - 现在更容易设定变数。

下面详细介绍 Keil C51 μ Vision4 IDE 基本的功能和使用。

第一章 建立第一个 Keil C51 程序-使用 C 语言

随着单芯片技术的不断发展，以单芯片 C 语言为主的高级语言 IDE，也不断的被开发出来，而且受到许多的单芯片爱好者和工程师所喜爱，更在学校中被广泛的使用在单芯片课程或微处理机课程教学上。Keil C51 μ Vision4 是众多单芯片 IDE 软件中优秀的软件之一，它支持许多不一样公司的 MCS-51 架构的芯片，它集编辑(Edit)，编译(Compiler)，模拟(Simulation)等于一体，同时还支持，PLM，汇编语言和 C 语言的程序设计，它的界面和微软的 VC++ 的界面相似，易学易用，在程序除错，软件仿真方面也有很强大的功能。使用 C51 写好 C 程序，然后用 C51 的编译程序把写好的 C 程序编译为机器码，这样单芯片才能执行编写好的 C 程序。

下面结合 MCS-51 介绍单芯片 C 语言的优越性：不须完全懂得单芯片的硬件架构，也能够编写出符合硬件实际的专业水平的程序。

· 不懂完全得单芯片的指令集，也能够编写单芯片程序。不同函数的数据实行覆盖，有效利用单芯片上有限的 RAM 空间。

· 提供 auto, static, 和 const 等存储类型和专门针对 8051 单芯片的 data, idata, pdata, data, 和 code 等存储类型，自动为变数合理地配置地址。

· C 语言提供复杂的数据类型（阵列(Array)、结构(Structure)、联合(Union)、枚举(Enumeration)、指标(Pointer)等），极大地增强了程序处理能力和零活性。

· 提供 small, compact, 和 large 等编译模式，以适应单芯片上内存的大小。中断服务程序的现场保护和恢复，中断向量表的填写，是直接与单芯片相关的，都是由 C 编译程序代办。程序具有坚固性：数据被破坏是导致程序执行异常的重要因素。C 语言对数据进行了许多专业性的处理，避免了执行中间不正常的破坏。提供常用的标准函数库，以供用户直接使用。

· 有严格的句法检查，错误很少。可方便地接受多种实用程序的服务：如单芯片上资源的初始化有专门的实用程序自动生成，简化用户程序设计，提升执行的安全性等等。表头档案(header)中定义、说明复杂数据类型和函数原型，有利于程序的移植和支持单芯片的系列化产品的开发。

以上简单介绍了 Keil C51 软件，要使用 Keil C51 软件，必需先要安装它，这也是学习单芯片 C 语言所要求的第一步的建立学习环境。

使用者可到 Keil C51 的官方网站下载(<https://www.keil.com/demo/eval/c51.htm>)，在网页中填妥个人资料，就可下载免费版，如图 1-1 所示。此免费版有 2K ROM 大小的限制，不过一般通常在学校使用，使用 2K ROM 的大小就绰绰有余了，若使用超过 2K ROM 的大小的话，就必须购买正式版了。安装方法很简单，安装时只要点选 C51V900.exe，就可自动执行安装了，其他后续版本也都一样，这里就不做介绍了。

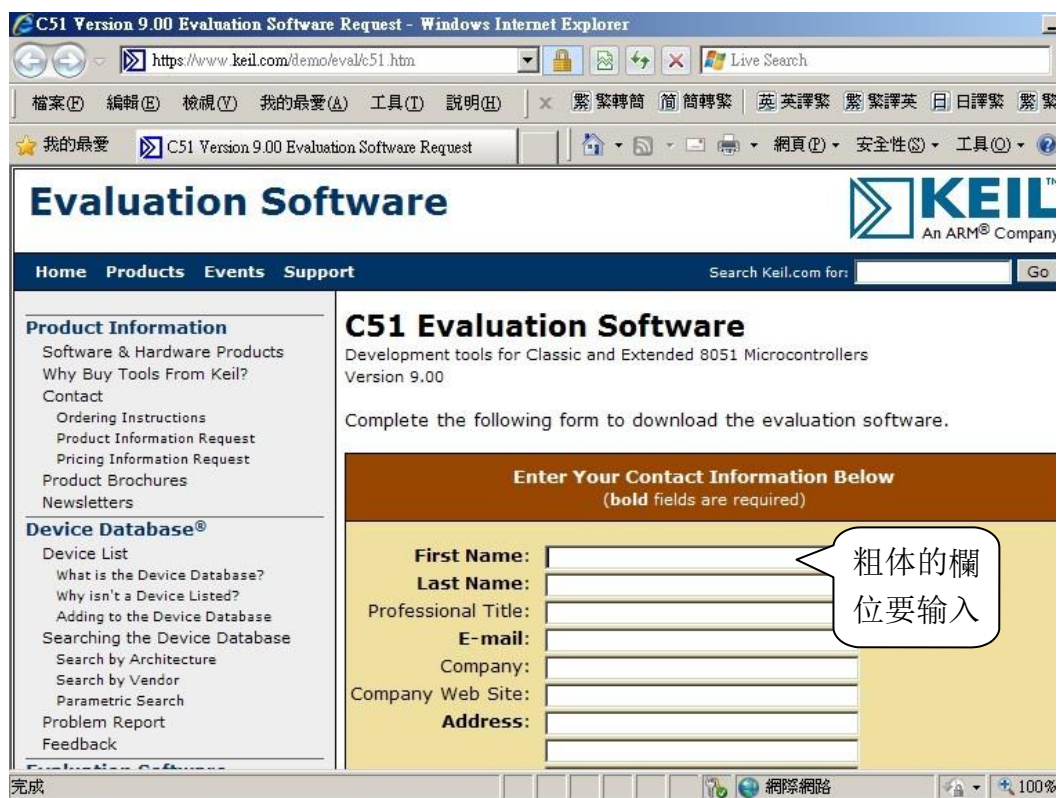


图 1-1 Keil C51 μ Vision4 官方下载网站
安装好 C51 后，C51 的初始主画面如图 1-2 所示。

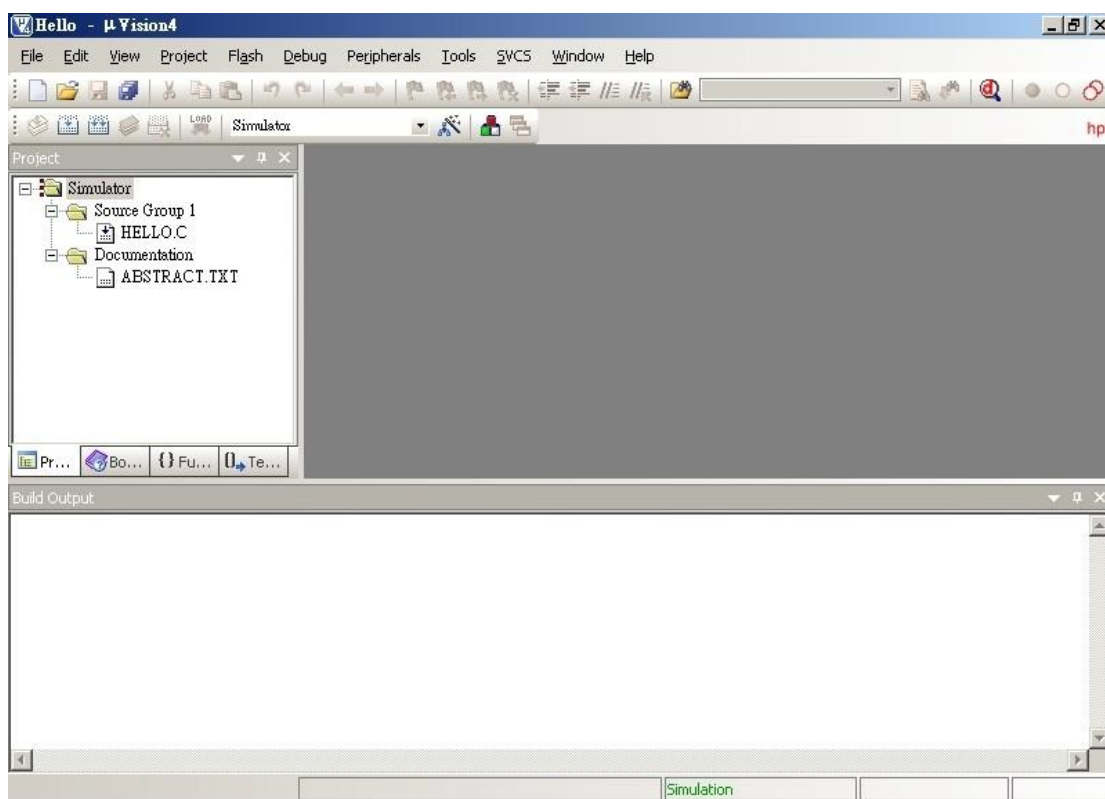


图 1-2 Keil C51 初始主画面 接着下面就让我们一起来建立自己的第一个单芯片 C 语言程序吧。请根据下面步骤

一步步的來，你绝对可以在短时间内熟悉 C51 的使用。

(1) 点击 Project(专案)选单，选择弹出的下拉式选单中的“New μ Vision Project...”，如图 1-3。接着弹出一个标准 Windows 档案对话窗口，如图 1-4。在“储存于”中选择您要存放的资料夹，一个项目最好存在一个资料夹内，若此资料夹不存在，请先建立它，或按“建立新资料夹”按钮以建立新资料夹。在“档名”中输入您的第一个项目名称，这里我们用“test1”。“存档类型”为 uvproj，这是 Keil μ Vision4 项目档案预设的扩展名，以后只要直接点击此项目文件，即可打开此项目。

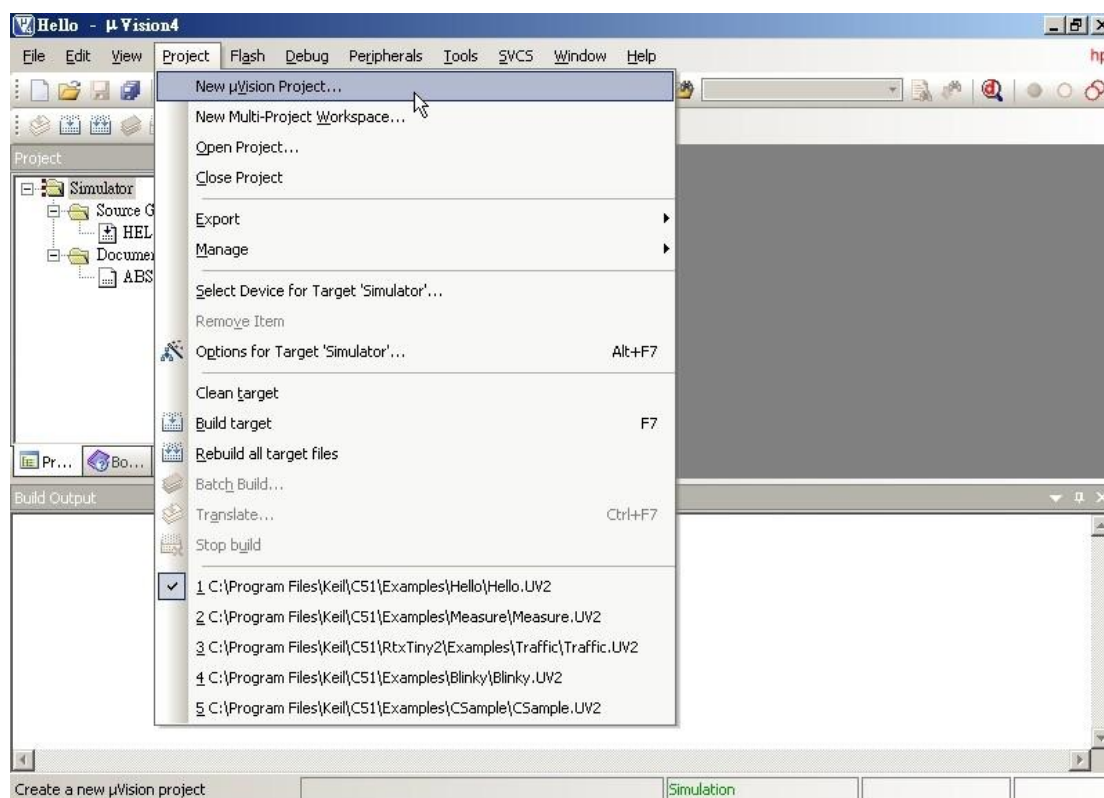


图 1-3 New μ Vision Project 选单

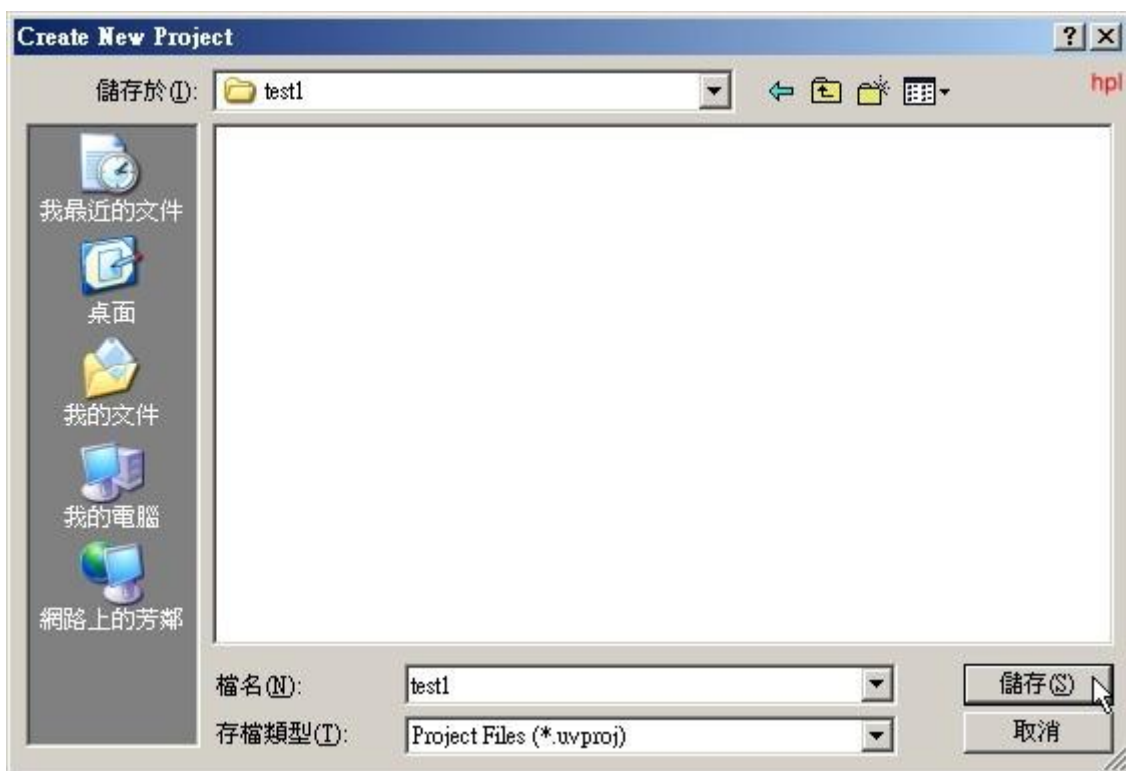


图 1-4 New μ Vision Project
选单

若第一次使用，则会出现图 1-5 的画面，若您要将您之前建立的旧版的项目档副档名更名为新版的项目档扩展名，则按确定，否则按取消，用户可根据自己的需求选择不变更或更名。因为第一次使用，按取消即可。

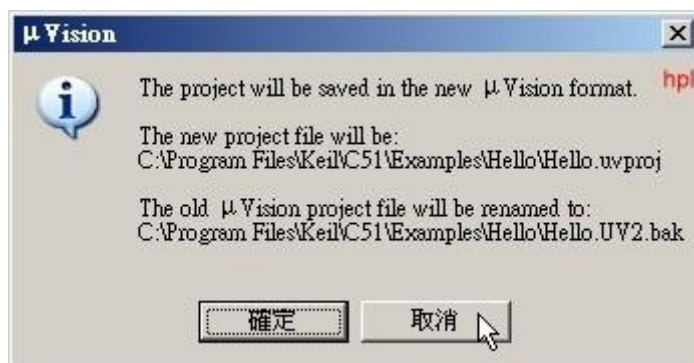


图 1-5 选择旧版或新版的专案档扩展名

(2) 选择所要的单芯片型号，这里选择常用的 Atmel 公司的 AT89S51。目前 Atmel 已经停产 AT89C51/AT89C52，改由 AT89S51/AT89S52 来取代之。AT89S 系列较 AT89C 系列更为便宜，工作频率可高达 33 MHz，且重复烧录次数至少可达一千次以上，非常适合学校学习或工程师开发产品之用。所以本讲义里的大部分程序都是基于此 AT89S51 芯片的，此时屏幕如图 1-6 所示。在右边图中的“Description”方块内，会简单的介绍 AT89S51 有什么功能及特点。点选 OK 按钮后，会出现图 1-7，询问你是否需要拷贝标准的 8051 启动码程序(STARTUP.a51)到你的项目资料夹，并且将此档案加入项目“Copy Standard 8051 Startup Code to Project Folder and Add File to Project”，点选“是”后，就可以进行程序的编写了。

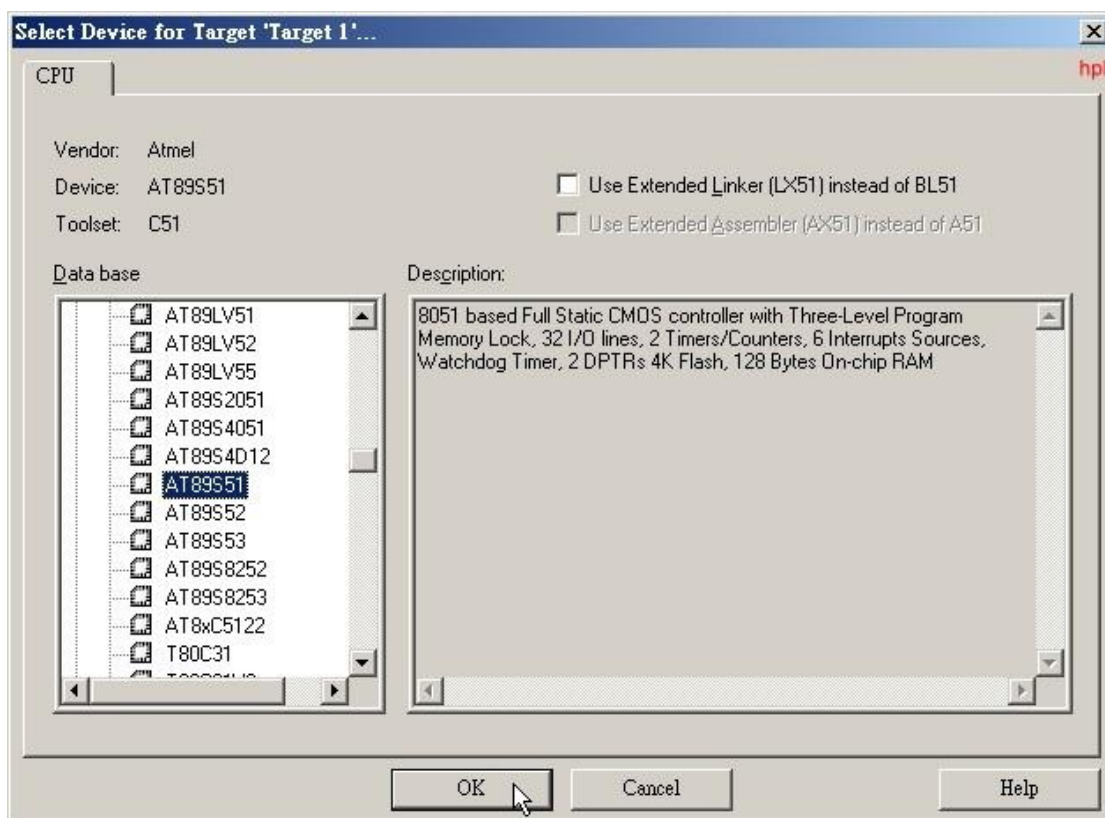


图 1-6 选取芯片
型号



图 1-7 询问是否需要加入 8051 启动码

STARTUP.a51 的主要工作是把包含 idata、xdata、及 pdata 在内的内存区块清除为

0，并且初始化递归指标。STARTUP.a51 的内容在附录 1 中，用户可自行参考。注意，若是编写汇编程序，则不需加入此启动程序。在完成上述的初始化程序之后，8051 的控制权才会交给 main() 主程序开始执行用户的程序。

(3) 首先在项目中建立新的程序档案或加入旧程序档案。如果您没有现成的程序或是第一次使用，那么就要新建一个 C 程序档案。在 C51 中有一些程序的范例，但是在这里我们还是以一个 C 程序为例介绍如何新建一个 C 程序，和如何加到您的第一个专案中吧。点击图 1-8 中 1 的新建文件的图标按钮，在 2 中出现一个新的文字编辑窗口，或是也能透过选单 File/New 或是按下快捷键 Ctrl+n 来实现。接着现在就能编写程序了。

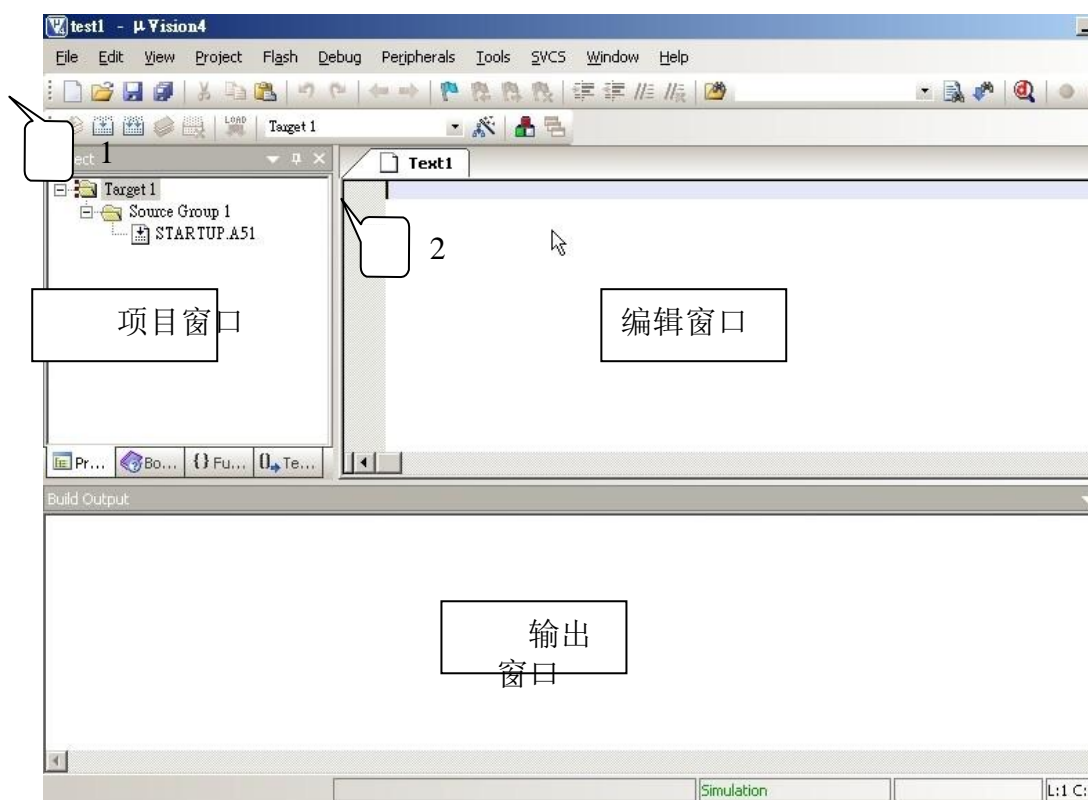


图 1-8 新建
文件

下面是一段一颗 LED 在闪烁的程序，在上图 2 的文件编辑工作区内键入下面的程式，先不管程序的语法和意思，先看看如何把它存档，加入到项目中存档，和如何编译及执行。AT89X51.h 的内容在附录 2 中，若你用的单芯片是 AT89S51，则用<AT89X52.h>。

```
#include <AT89X51.h>
main()
{
    int i;
    while(1)
    {
        P0_0=1;
        for(i=1;i<20000;i++);
        P0_0=0;
        for(i=1;i<20000;i++);
    }
}
```

(4) 点击图 1-9 中的储存档案图标按钮，也能用选单 File/Save 或按快捷键 Ctrl+S，

则出现图 1-10 的窗口。把此程序命名为 test1.c，储存在项目所在的资料夹中，再按储存钮。这个时候您会发现程序单字有了不同的颜色，这表示 Keil 的 C 语言语法检查开始作用了。

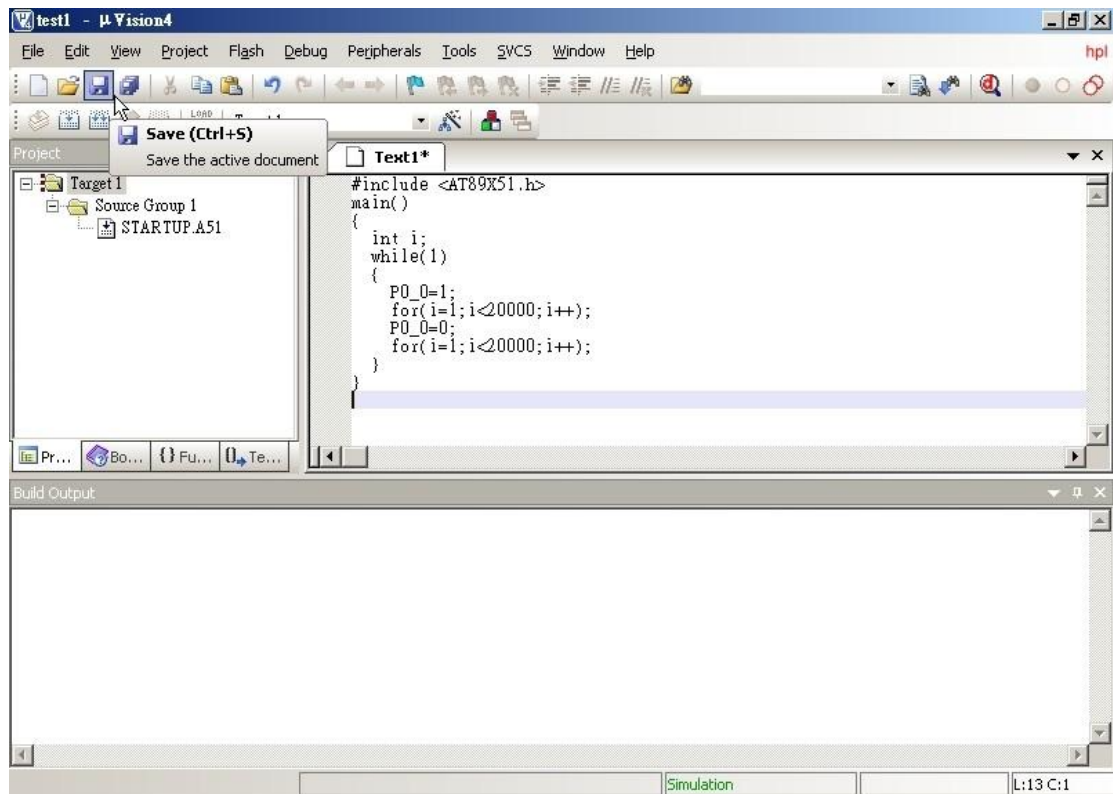


图 1-9 储存



图 1-10 储存 c
档案

(5) 滑鼠在屏幕左边的 Source Group1 资料夹图示上右键单击，弹出一选单，如图 1-11 所示，在这里能做在项目中增加减少档案等操作。选 “Add Files to Group ‘Source Group 1...’ ” 弹出档案窗口，选择刚刚储存的档案，按下 Add 按钮，将此 c 档案加入到

此专案中。按下 close 按钮，关闭档案窗口，如图 1-12 所示，则此 test1.c 程序档案已加 到此项目中了，如图 1-13 所示。

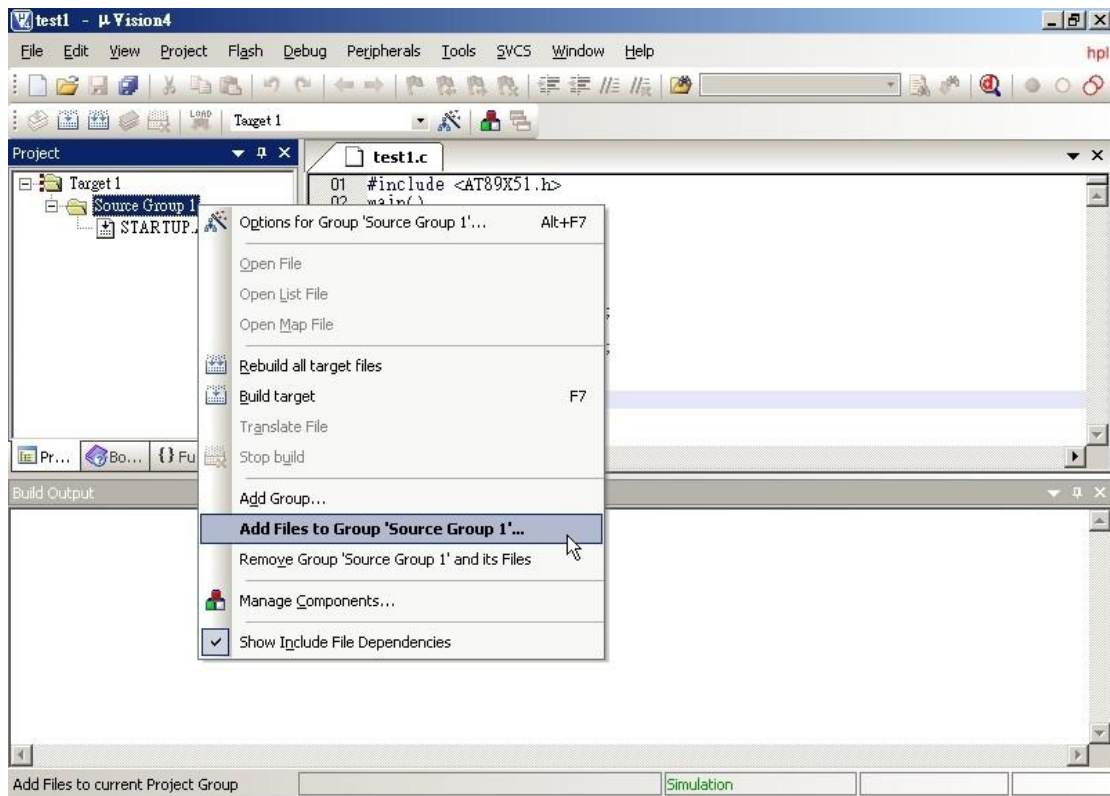


图 1-11 选取 “Add Files to Group ‘Source Group 1’...”

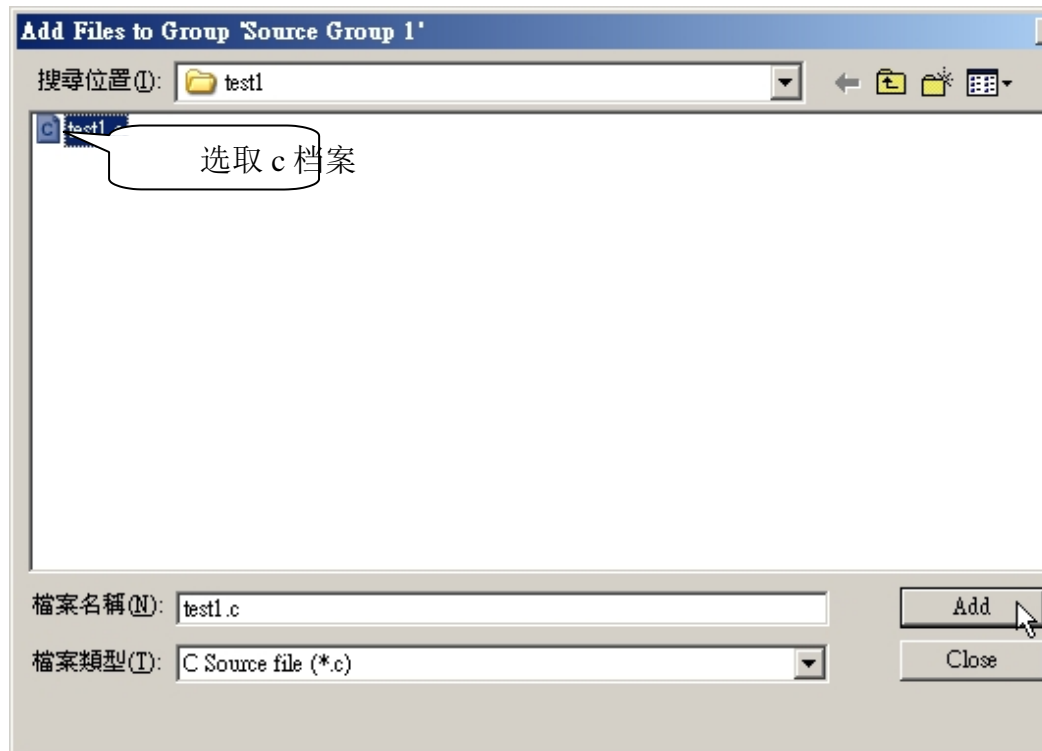


图 1-12 选取要加入到项目中的 c 档案

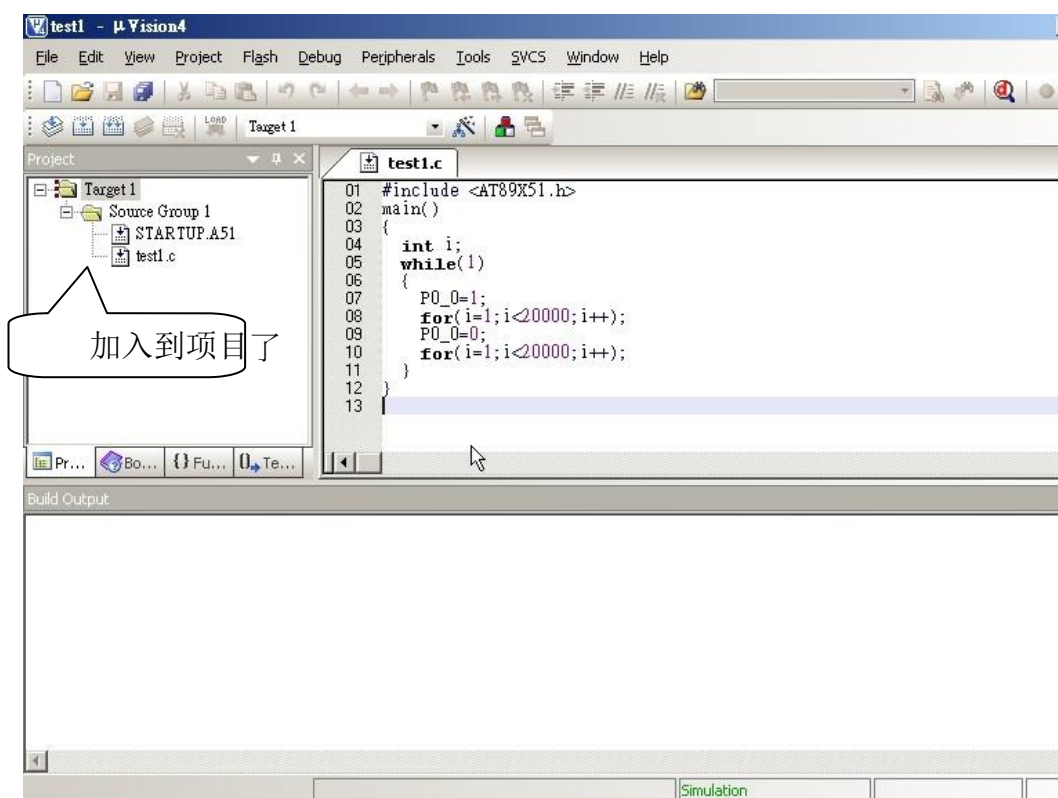


图 1-13 把 c 档案加入到项目中了 若用户写的是汇编语言档案，那必须存成 *.a51 或 *.asm 档，然后将汇编语言档案加入到专案中。

(6) 接下来要来做一些基本的设定选项的工作。滑鼠在屏幕左边的 Target 1 资料夹图示上右键单击，弹出一选单，如图 1-14 所示，然后选取“Options for target ‘Target 1’...””。

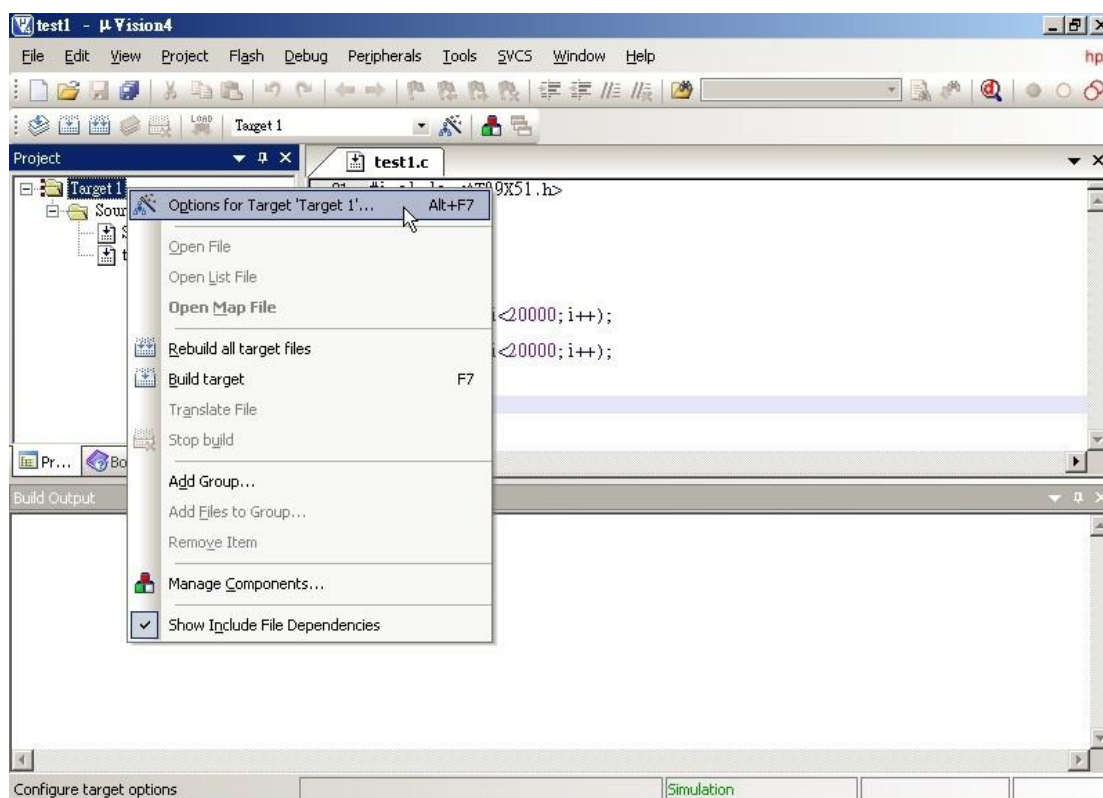


图 1-14 选取“Options for target ‘Target 1’...”

在图 1-15 的 Target 标签页中，更改所选用单芯片的工作频率为 12。MCS-51 系列单芯片系统一般常选用 11.059 MHz 或 12 MHz。前者适用于产生各种标准的鲍率(baud rate)，后者的一个机器周期为 1 μ s，便于产生精确延迟时间。本程序中假设使用频率为 12 MHz 的晶体振荡频率。另外，勾选 Use On-Chip ROM(0x0-0xFFFF)，以使用单芯片上的 Flash ROM。

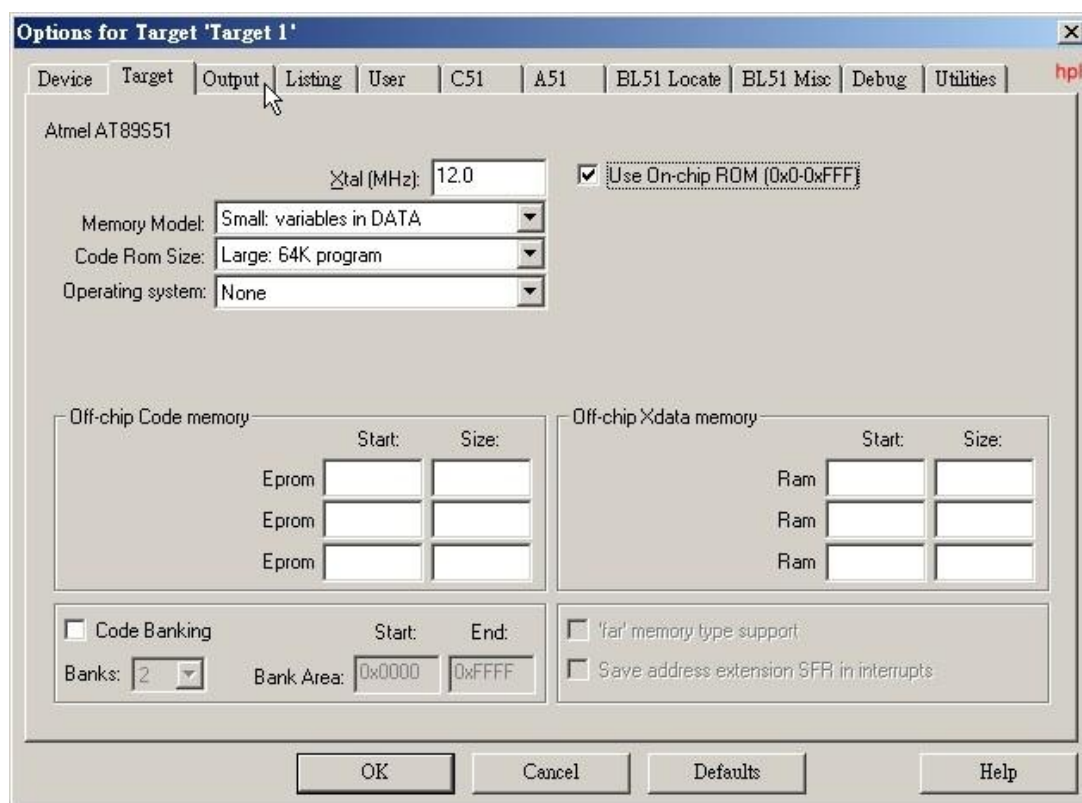


图 1-15 在 Target 标签页中，更改所选用单芯片的工作频率

再来切换到 Output 标签页，只要勾选“Create HEX File”就好了，以产生烧录档，如图 2-13 所示。如果用户只是单纯的做练习，那就省略此步骤了。若要更改存放目的档的资料夹，则点击“Select Folder for Objects...”，若要更改编译后的主档名，则在“Name of Executable:”右边的空格内输入主档名即可，一般而言，这 2 个选项都采用默认值，用户不需更改他们。

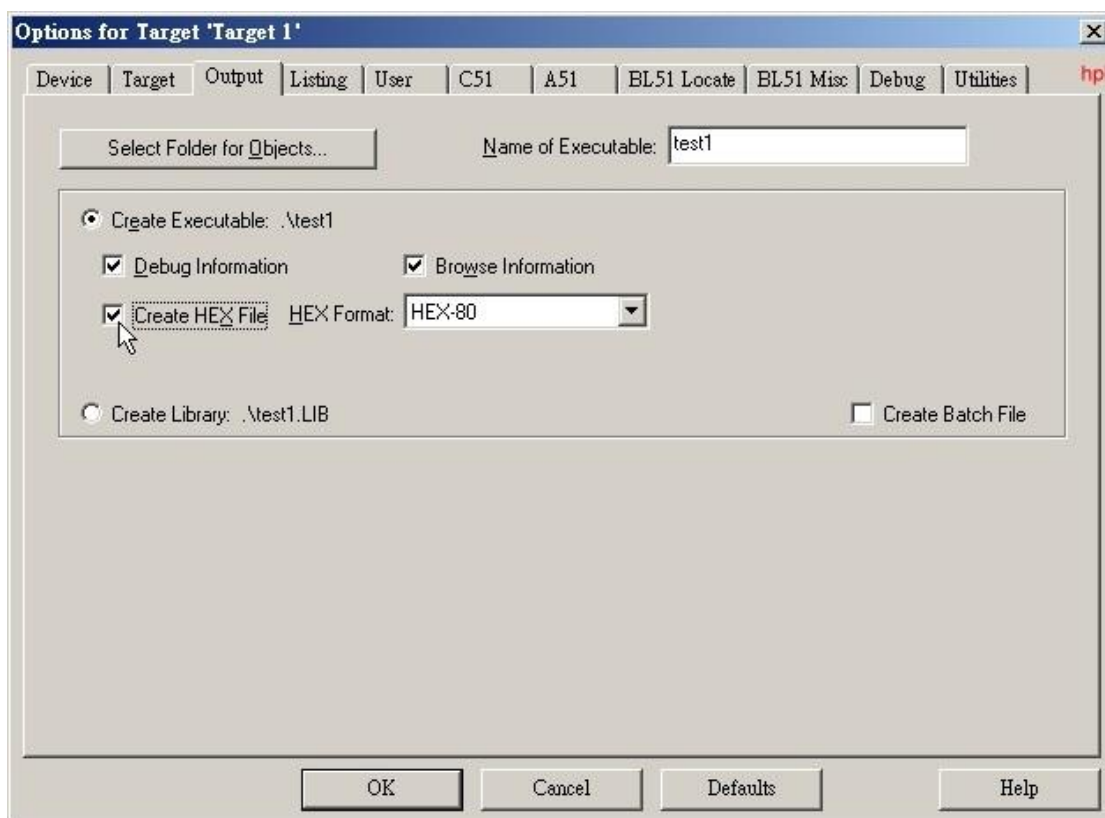






图 1-16 勾选“Create HEX File”，以产生烧录档

(7) 完成基本的选项设定后，下面就剩下编译执行了。先来看这三个按钮



这三个都是编译按钮。按钮是用于编译目前工作区的档案但不做连结(Link)，按钮是用于编译整个项目文件并连结，如果之前编译过一次之后档案没有做任何编辑的话，

这个时候再点击是不会再次重新编译的。按钮是重新编译整个项目文件并连结，每点

击一次均会再次编译连结一次，不管程序是否有改变。是停止编译按钮，只有点击了前三个中的任一个，停止按钮才会生效。或是从选单 **Project** 中，也可执行编译，在此

按下“Build target”或 F7 快捷键，如图 1-17 所示。

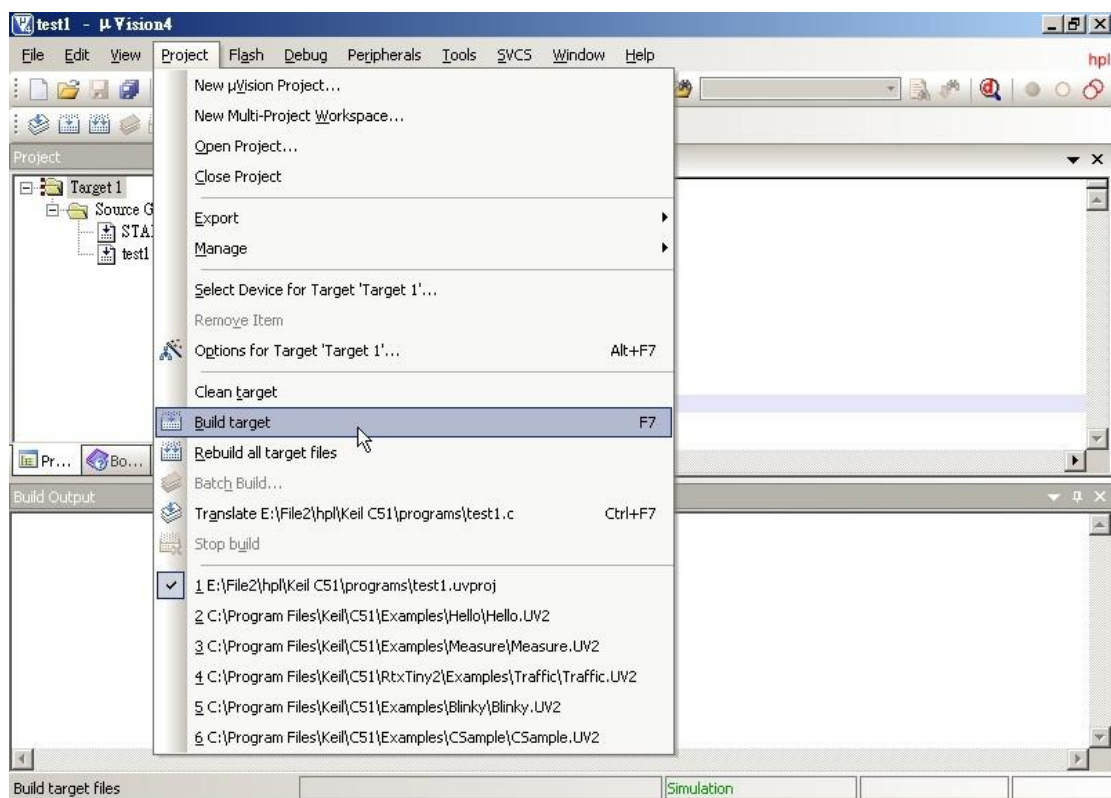


图 1-17 编译项目

编译完成后，在下方的 Build Output 区域中，可看到编译的讯息，如图 1-18 所示。若有出现错误讯息，则再根据错误讯息，回到程序中修改，编译完全正确后，才能产生正确的烧录档 test1.hex。

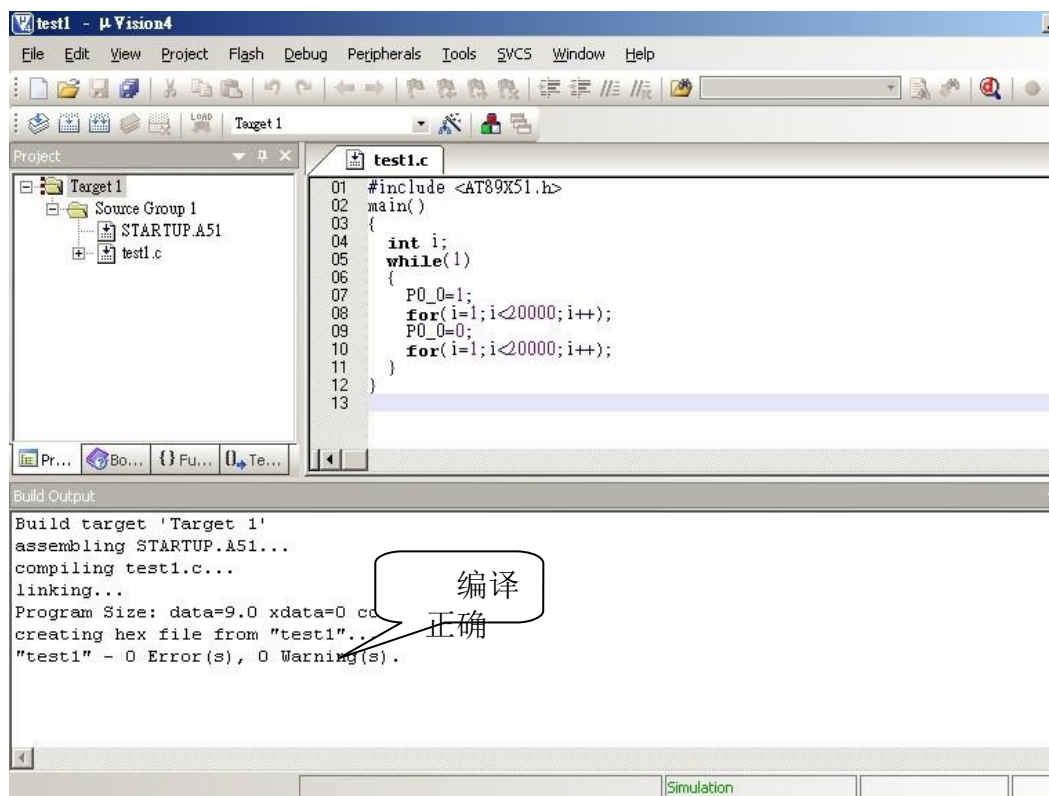








图 1-18 项目编译
正确

(8) 选取选单 Debug/Start/Stop Debug Session，或按快捷键 Ctrl+F5，或按在档案工具列右边有一个小放大镜的按钮，则进入除错(Debug)模式，并显示不同的工作窗口，如图 1-19 所示。进入除错模式之前，会先出现一个小窗口，告诉你目前用的版本是免

费的评估版，有 2K ROM 大小的限制，点击确定即可进入除错模式。在除错工具列中，

 (Reset)按钮表示重置单芯片，并使程序回到最开头处执行。 (Run)按钮表示执行， (Stop)按钮表示停止，当程序处于执行状态时，停止按钮才有效。 (Step Into)按钮表示单步执行会进入函数内，(Step Over)按钮表示单步执行不会进入函数内，(Step Out)按钮表示离开函数， (Run to Cursor)按钮表示执行到光标所在处。

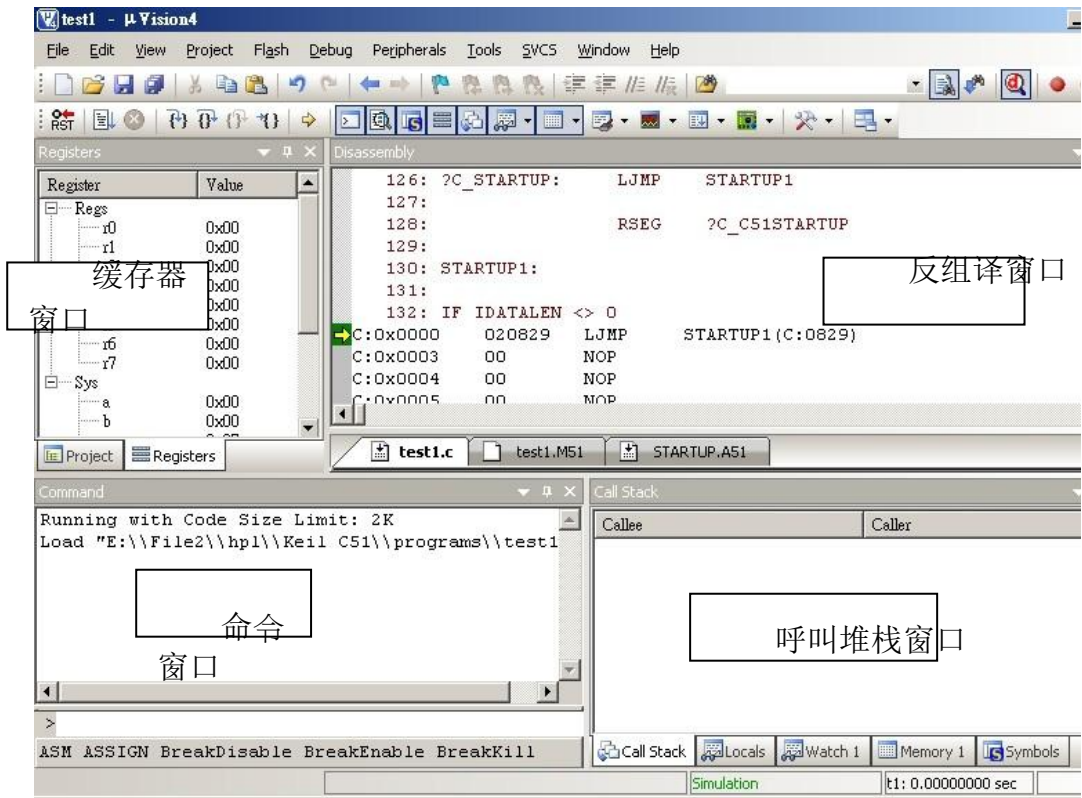


图 1-19 程序除错
画面

为了要检视输出结果是否正确，则必须叫出P0输出埠观察输出结果。选取选单 Peripherals/I/O-Ports/Port 0，如图1-20所示。

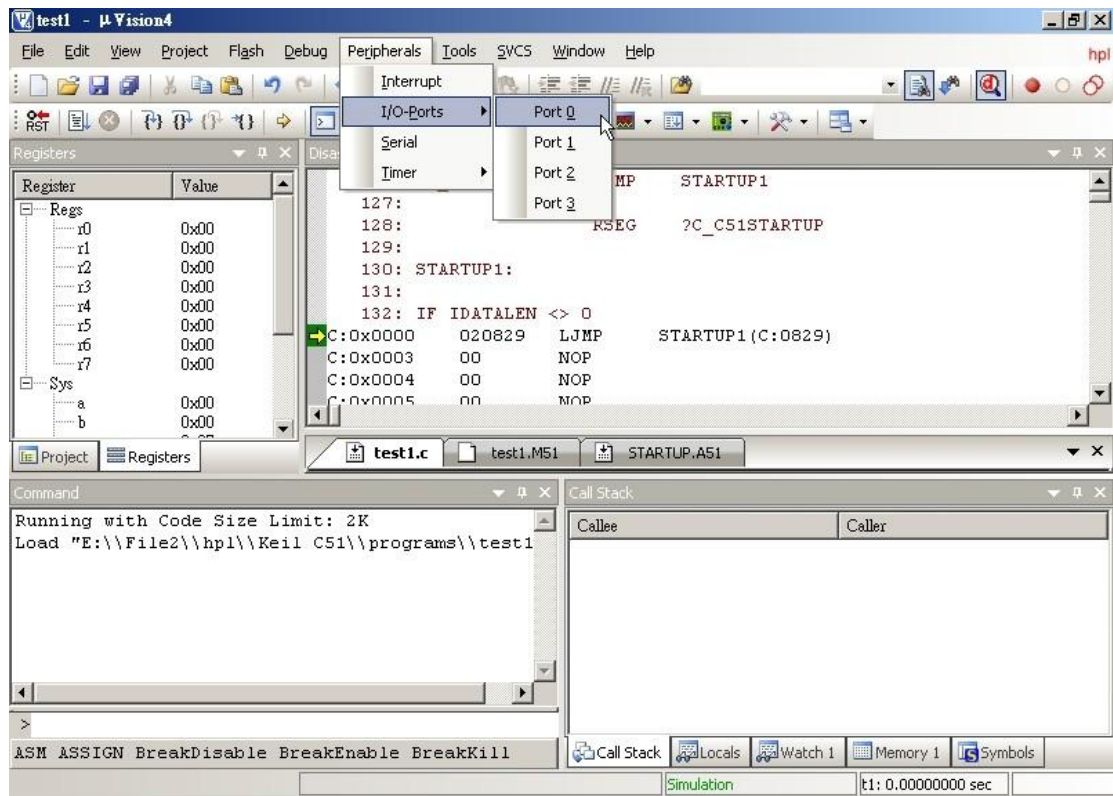


图 1-20 选取
Port 0

出现 Parallel Port 0 小窗口，并显示每一个位的值，也可移动到其他位置观察，如图 1-21 所示。

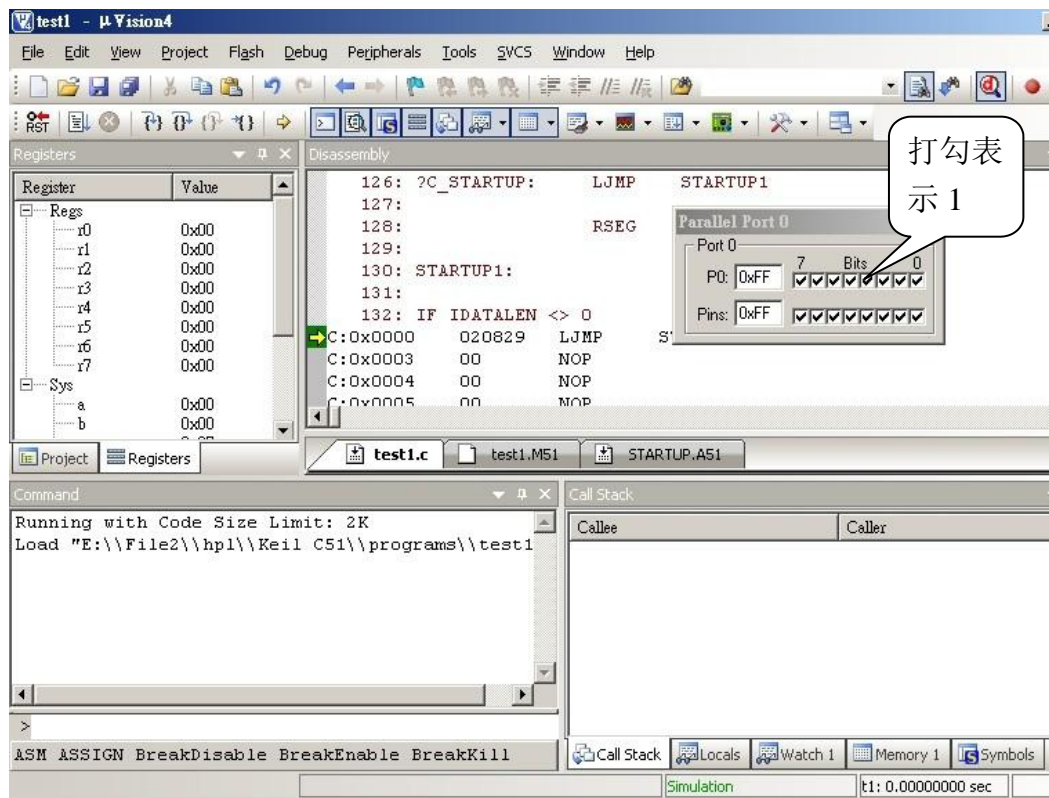




图 1-21 显示 Port 0 的
窗口

最后要准备执行此程序了，先单击重置  按钮，让单芯片及程序回到最初状态，再按下执行  按钮后，则程序开始执行。我们可以看到 Parallel Port 0 窗口中的 P0_0 位元不断的被设定与清除。

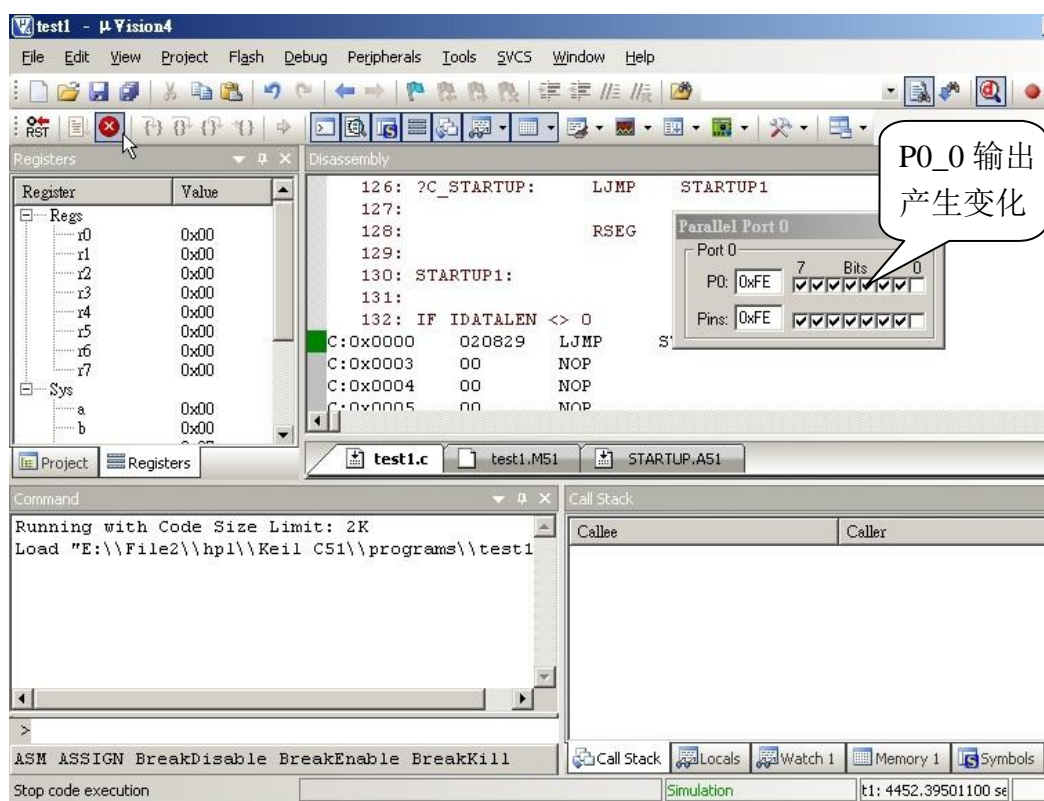


图 1-22 显示 Port 0 的内容及停止程序执行 若要观察汇编程序编译后所产生的运算码，在 ROM 的存放情形，则选取选单

View/Memory Windows/Memory 1，或按右下方的 Memory 1 按钮，然后在 Address 欄位内输入 0x800 或 0800h，则 Memory 1 窗口从 0x0800 开始显示运算码，如图 1-23 所示。

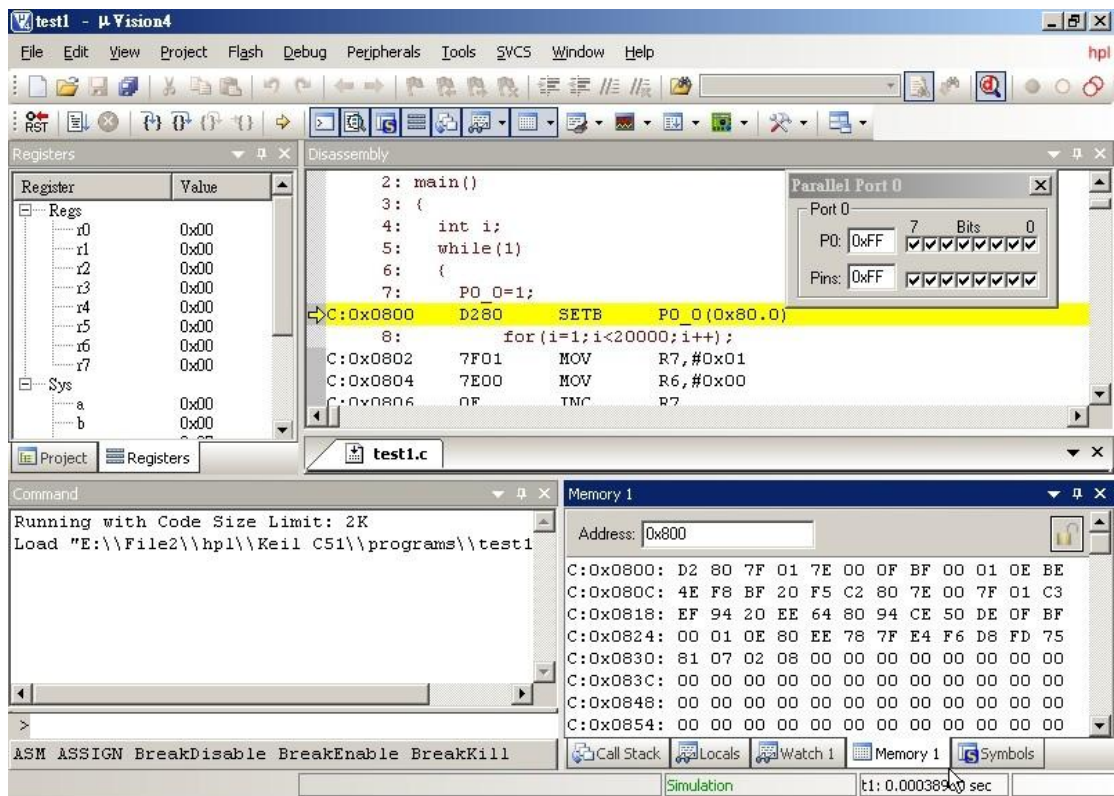




图 1-23 Memory 1 窗口显示运算码

程序执行正确后，最后要停止程序执行回到档案编辑模式中，就要先按停止  按钮再按开启/关闭除错模式  按钮。若要关闭此专案，则选取选单 Project/Close Project，关闭此专案。Keil C51 μVision4 的 C 语言程序初次使用，到此告一段落，下一章我们看 Keil C51 μVision4 的汇编程序的初次使用。

第二章 建立第二个 Keil C51 程序-使用汇编语言

虽然以 C 语言或以汇编语言撰写 MCS-51 程序來相比，C 语言在功能上、架构性、可读性、可维护性上有明显的优势，因而易学易用。然而汇编语言所写出来的程序经编译(compiler)后，所产生出来的运算码或机械语言码所占用的内存，会比 C 语言所写出来的程序经编译后，所产生出来的运算码所占用的内存少，因此执行效率较高。C 语言要经过编译后转成汇编语言，而转出来的汇编语言的内容的写法有时后让人觉得有划蛇添足的感觉。然而对程序如果要求要写得简单、直接、有效率的话，通常是会选择汇编语言。一般而言，对读电子电机或资工系的学生，最好是 C 语言和汇编语言都要学。

接着下面就让我们一起来建立自己的第二个单芯片汇编程序吧。若您对上一章的操作已经有了初步认识的话，那你对下面的步骤，就会觉得更容易。

(1) 点击 Project(专案)选单，选择弹出的下拉式选单中的“New μ Vision Project...”，如图 2-1。接着弹出一个标准 Windows 档案对话窗口，如图 2-2。在“储存于”中选择您要存放的资料夹，一个项目最好存在一个资料夹内，若此资料夹不存在，请先建立它，或按“建立新资料夹”按钮以建立新资料夹。在“档名”中输入您的第二个项目名称，这里我们用“test2”。“存档类型”为 uvproj，这是 Keil μ Vision4 项目档案预设的扩展名，以后只要直接点击此项目文件，即可打开此项目。

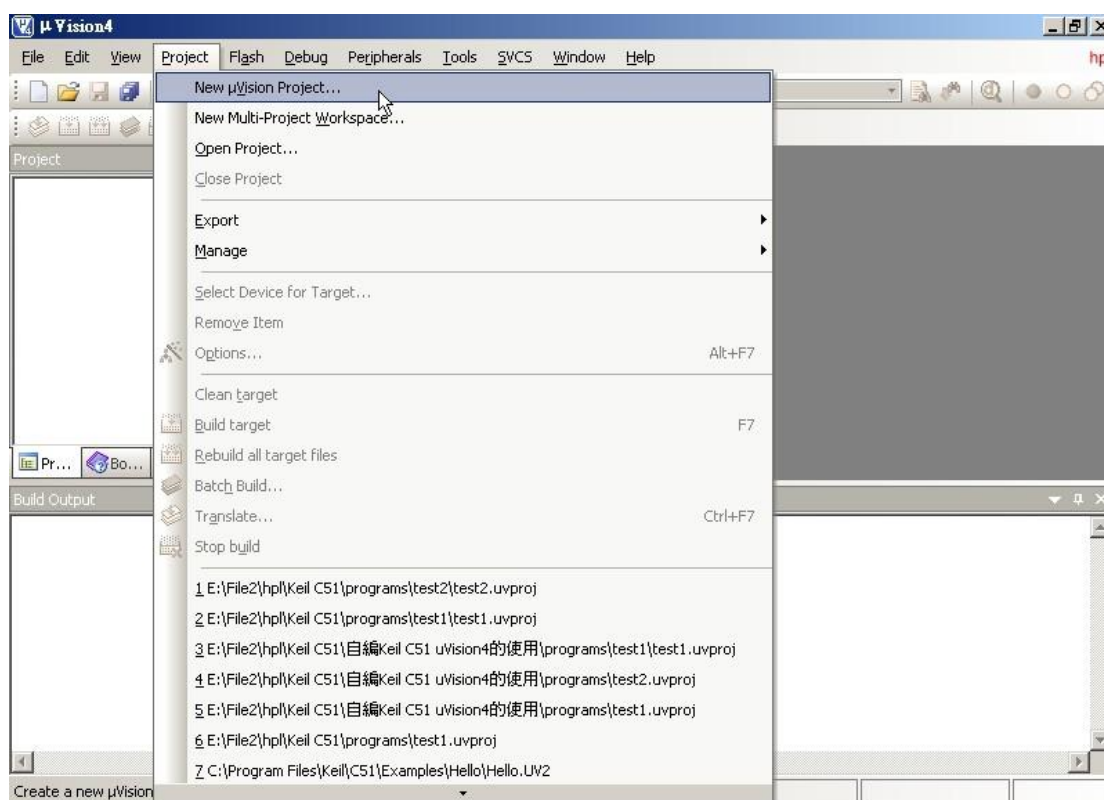


图 2-1 New μ Vision Project 选单

图 2-2 New μ Vision Project 选单

因第二次使用，所以不会出现图 1-5 的画面。

(2) 选择所要的单芯片型号，这里仍然选择常用的 Atemp 公司的 AT89S51，此时萤幕如图 2-3 所示。在右边图中的“Description”方块内，会简单的介绍 AT89S51 有什么功能及特点。点选 OK 按钮后，会出现图 2-4，询问你是否需要拷贝标准的 8051 启动码程式到你的项目资料夹，并且将此档案加入项目“Copy Standard 8051 Startup Code to Project Folder and Add File to Project”，因为汇编语言不需要加入此 STARTUP.a51，点选“否”后，就可以进行程序的编写了。

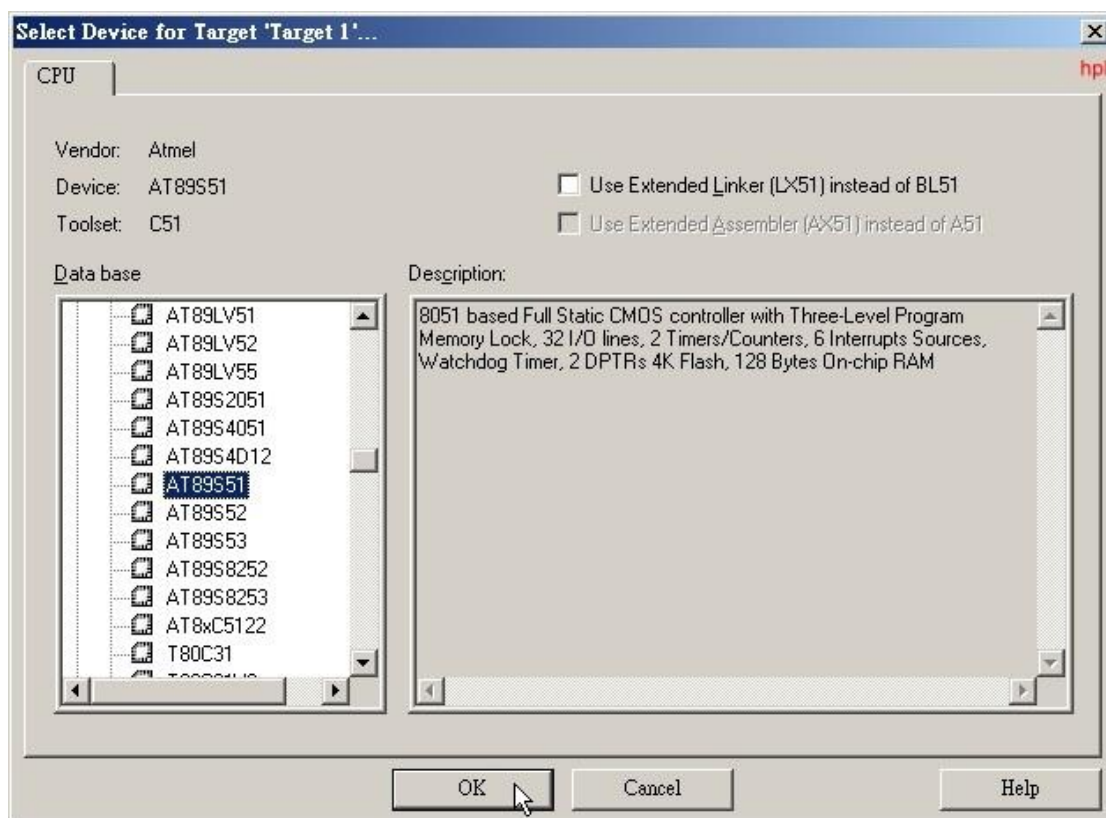


图 2-3 选取芯片
型号



图 2-4 询问是否需要加入 8051 启动码

(3) 首先在项目中建立新的程序档案或加入旧程序档案。如果您没有现成的程序或是第一次使用汇编语言，那么就要新建一个空白程序档案。在 C51 中有一些程序的范例，但是在这里我们还是以一个组合程序为例介绍如何新建一个组合程序，和如何加到您的项目中吧。点击图 2-5 中 1 的新建文件的图标按钮，在 2 中出现一个新的文字编辑窗口，或是也能透过选单 File/New 或是按下快捷键 Ctrl+n 来实现。接着现在就能编写程序了。

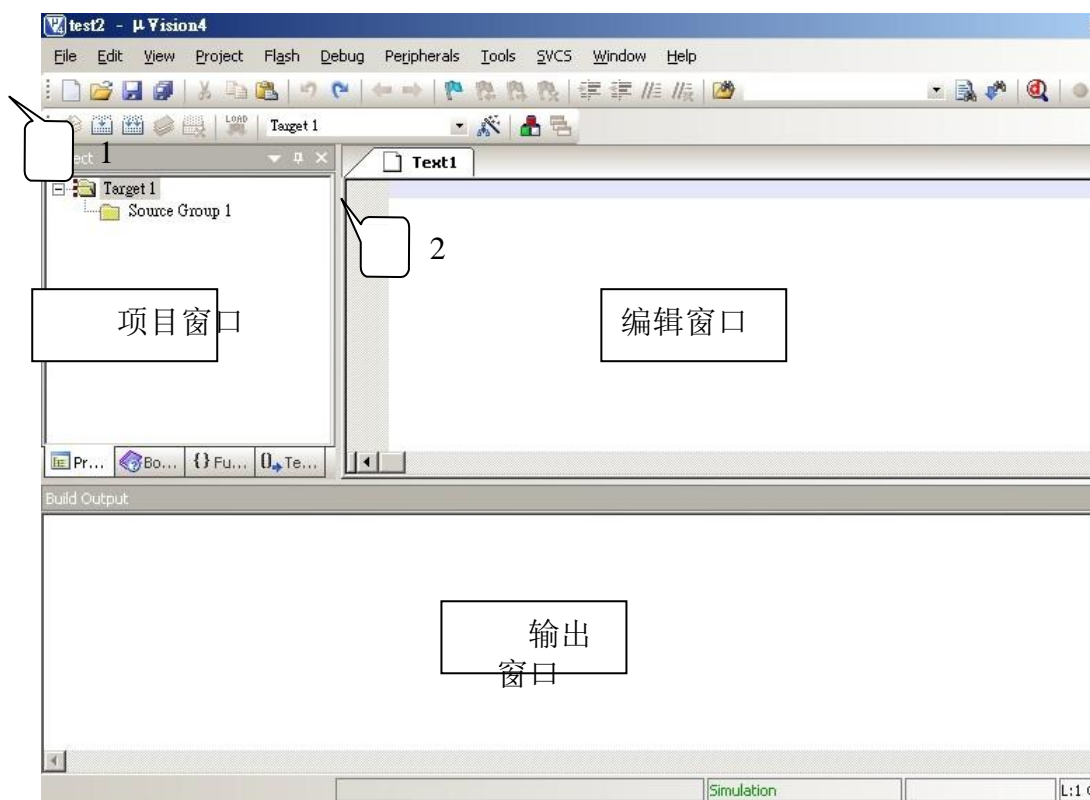


图 2-5 新建文件

下面是一个跑马灯的程序，在上图 2 的文件编辑工作区内键入下面的程序，先不管程序的语法和意思，先看看如何把它存档，加入到项目中存档，和如何编译及执行。

```

                MO      A,#0
STA             MO      P1,
                RL      A
                ACA     DEL
                AJM     STA
DEL            MO      R0,#
D1:            MO      R1,#
D2:            DJN     R1,
                DJN     R0,
                RET
                END
    
```

(4) 点击图 2-6 中的储存档案图标按钮，也能用选单 File/Save 或按快捷键 Ctrl+S，

则出现图 2-7 的窗口。把此程序命名为 test2.a51，储存在项目所在的资料夹中，再按储存钮。这个时候您会发现程序单字有了不同的颜色，这表示 Keil 的汇编语言语法检查开始作用了。

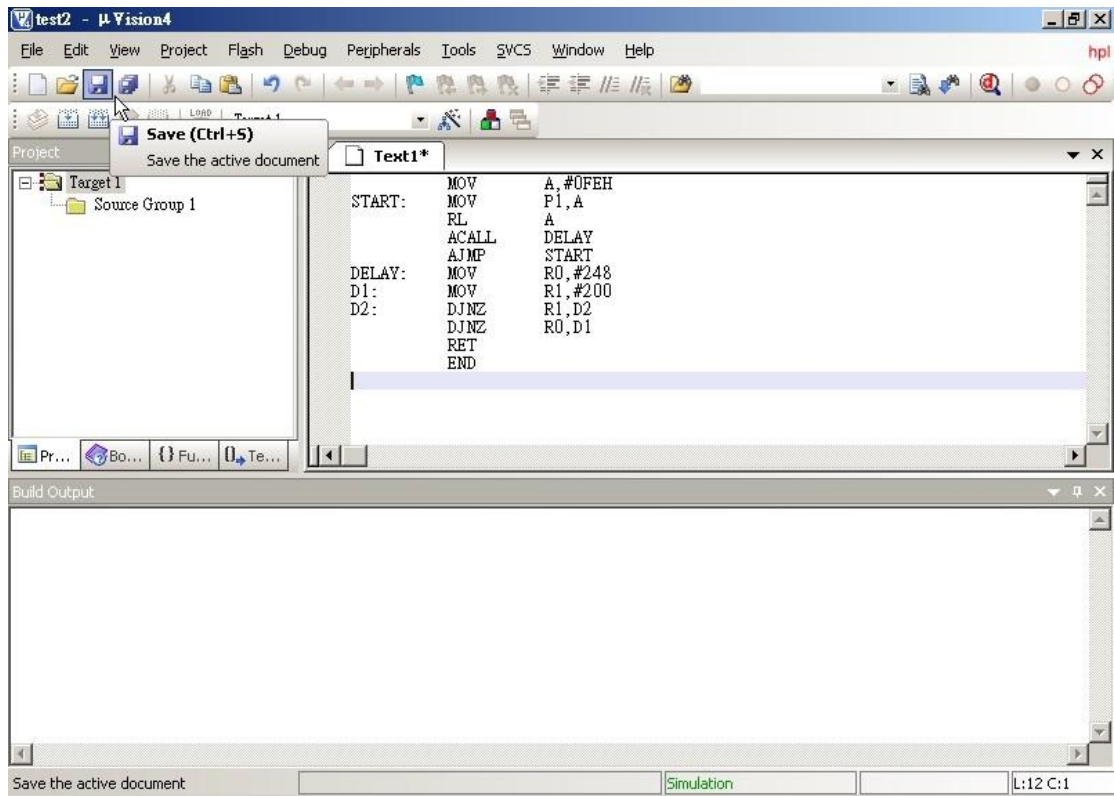


图 2-6 储存
档案



图 2-7 储存 a51
档案

(5) 滑鼠在屏幕左边的 Source Group1 资料夹图示上右键单击，弹出一选单，如图 2-8 所示，在这里能做在项目中增加减少档案等操作。选 “Add Files to Group ‘Source Group 1...’ ” 弹出档案窗口，选择刚刚储存的档案，按下 Add 按钮，将此.a51 档案加入

到此专案中。按下 close 按钮，关闭档案窗口，如图 2-9 所示，则此 test2.a51 程序档案已加到此项目中了，如图 2-10 所示。

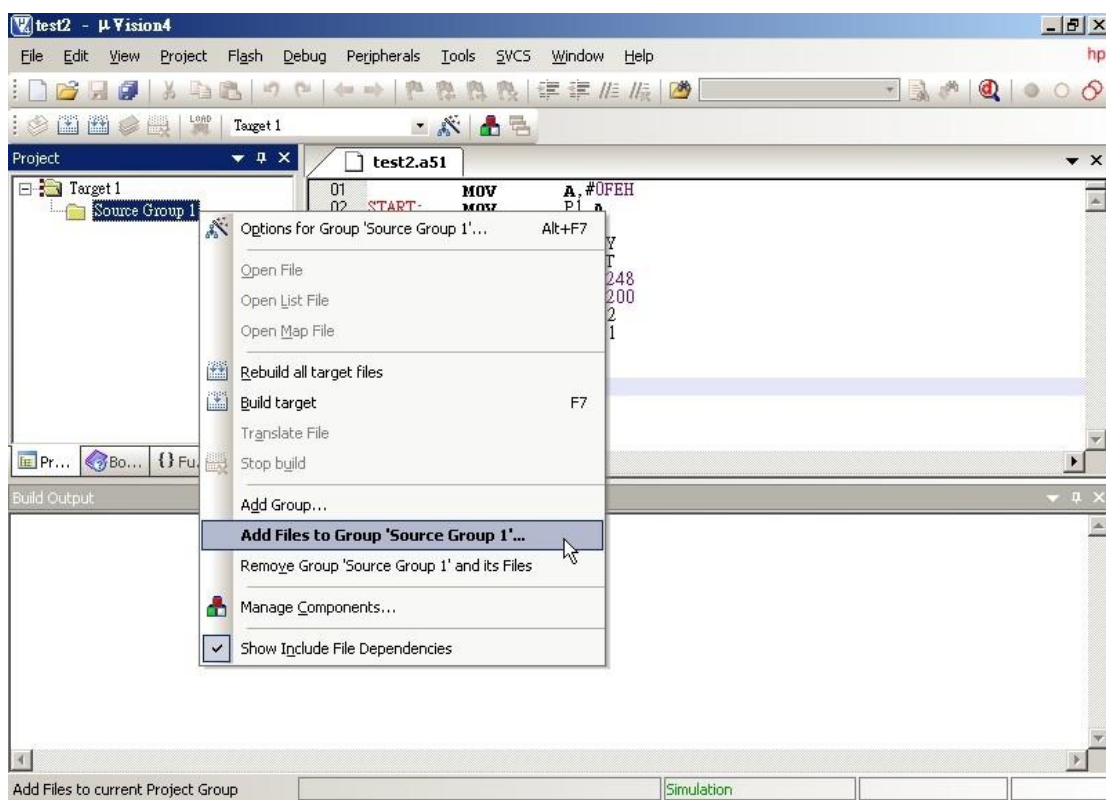


图 2-8 选取 “Add Files to Group ‘Source Group 1’...”

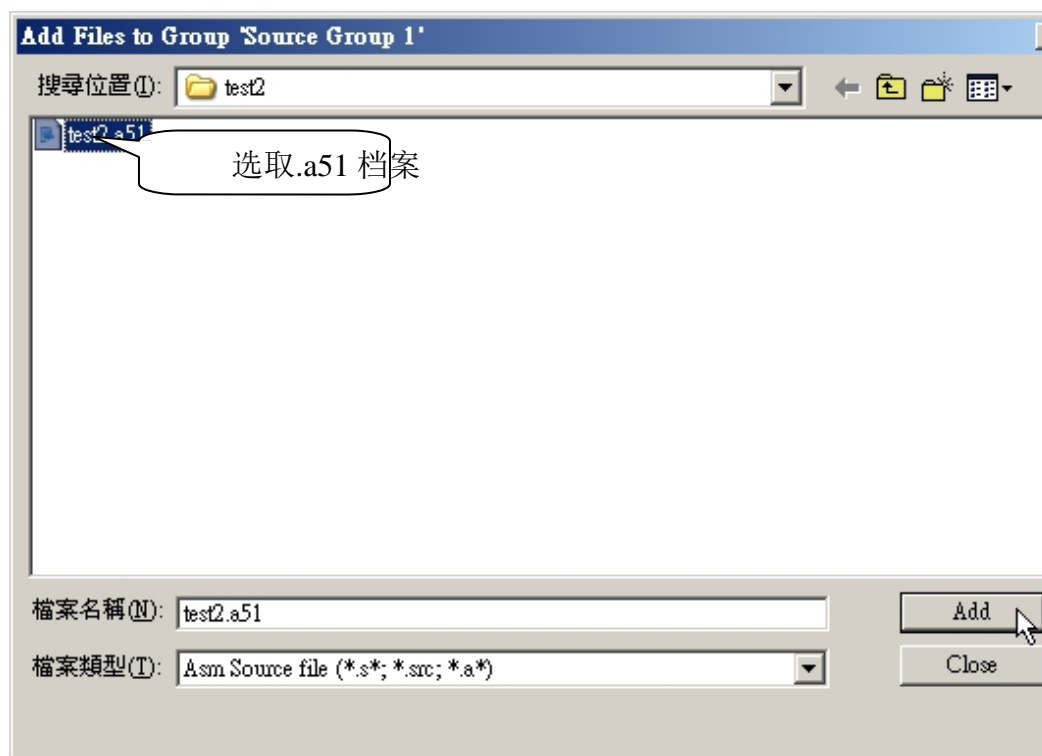


图 2-9 选取要加入到项目中的汇编语言档案

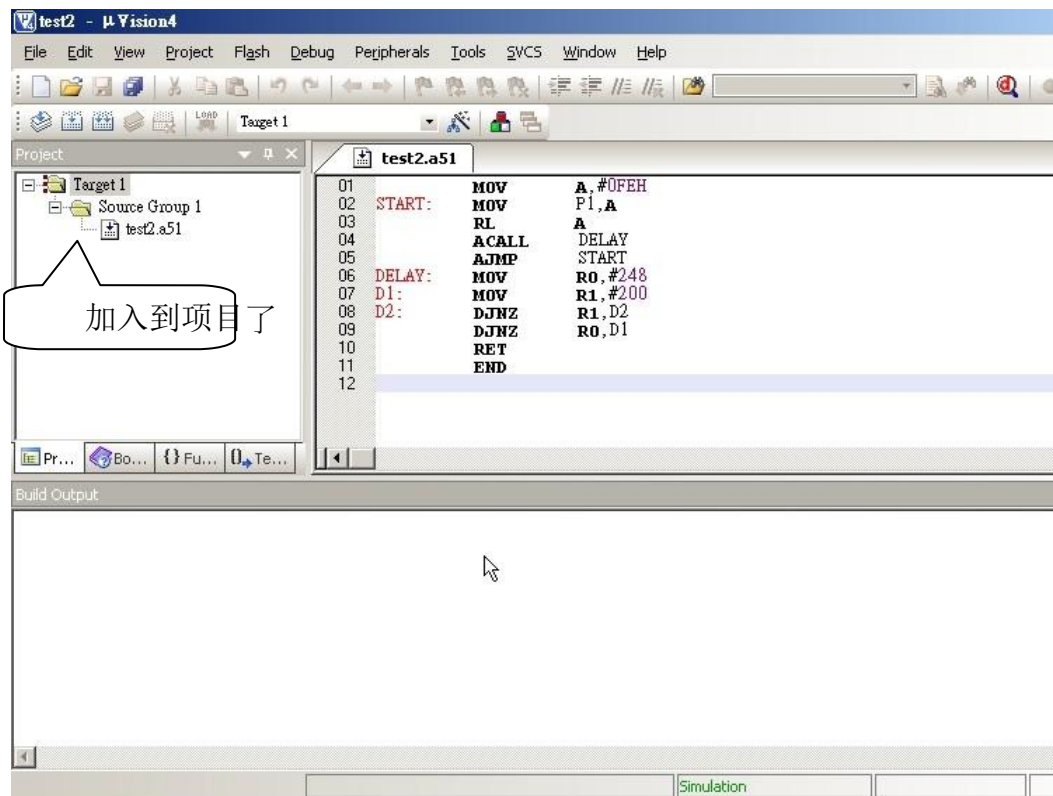


图 2-10 把汇编语言档案加入到项目中了

(6) 接下来要来做一些基本的设定选项的工作，在每一个项目里面都要设定一次。滑鼠在屏幕左边的 Target 1 资料夹图示上右键单击，弹出一选单，如图 2-11 所示，然后选取“Options for target ‘Target 1’...”，则出现图 2-12 的窗口。

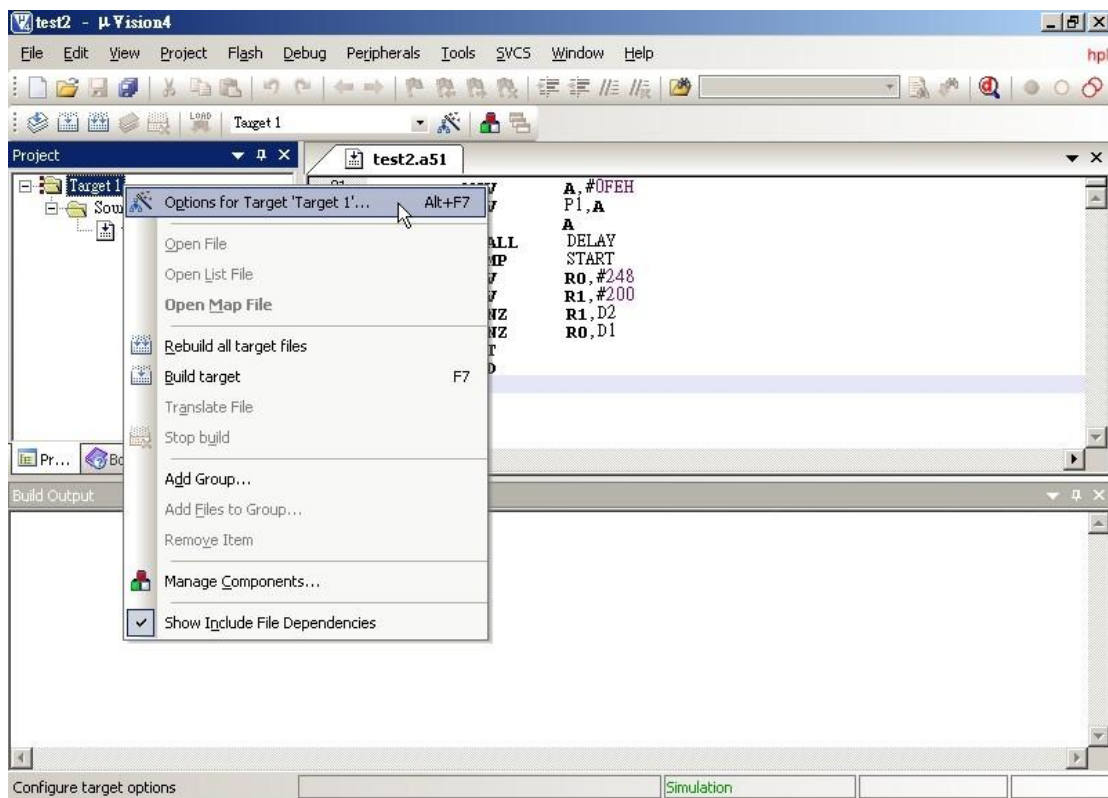


图 2-11 选取“Options for target ‘Target 1’...”

在图 2-12 的 Target 标签页中，如同图 1-15，更改所选用单芯片的工作频率为 12，并勾选 Use On-Chip ROM(0x0-0xFFFF)，以使用单芯片上的 Flash ROM。

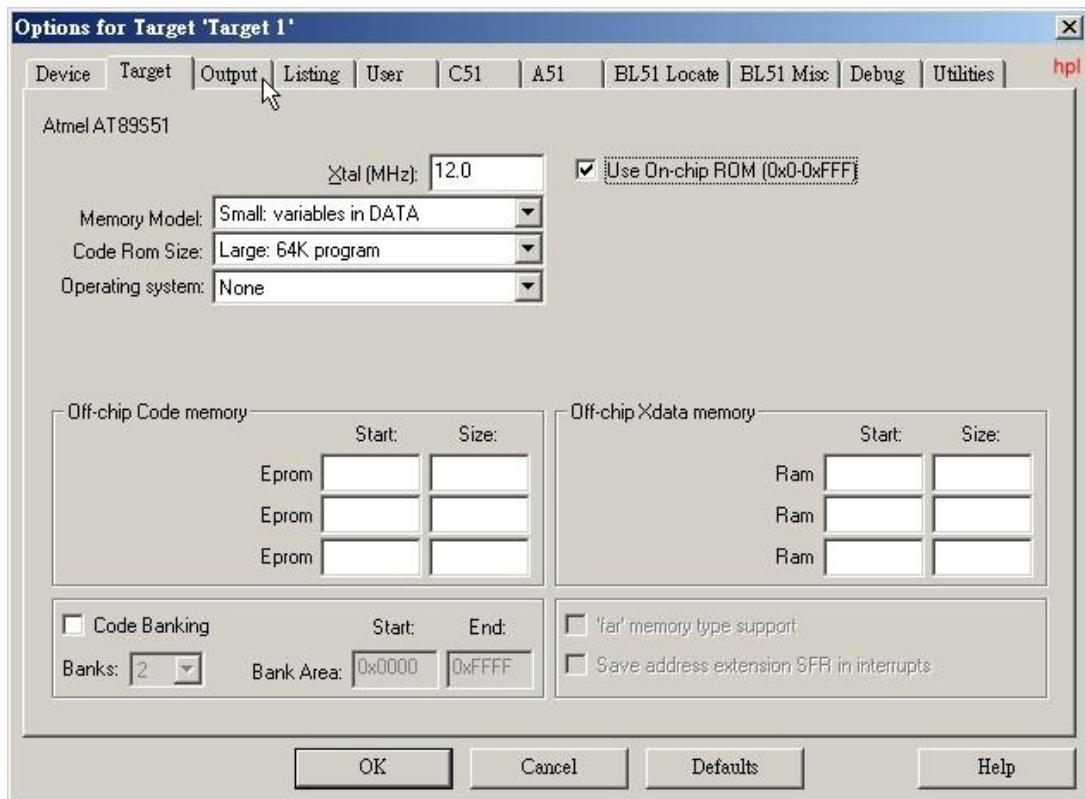


图 2-12 在 Target 标签页中，更改所选用单芯片的工作频率

再來切换到 Output 标签页，只要勾选“Create HEX File”就好了，以产生烧录檔，如图 2-13 所示。如果用户只是单纯的做练习，那就省略此步骤了。若要更改存放目的檔的资料夹，则点击“Select Folder for Objects...”，若要更改编译后的主档名，则在“Name of Executable:”右边的空格内输入主档名即可，一般而言，这 2 个选项都采用默认值，用户不需更改他们。

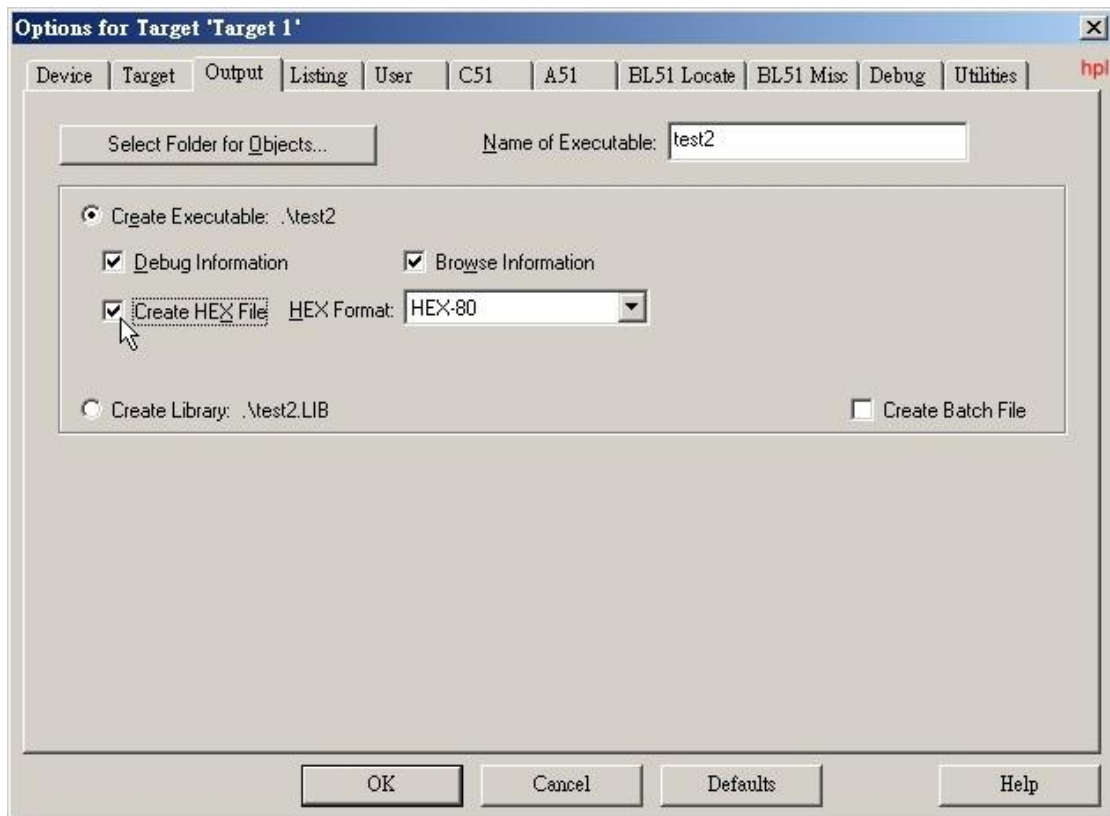



图 2-13 勾选“Create HEX File”，以产生烧录档

(7) 完成基本的选项设定后，下面就剩下编译执行了。在此按下  按钮，“Build target”，或 F7 快捷键，如图 2-14 所示。

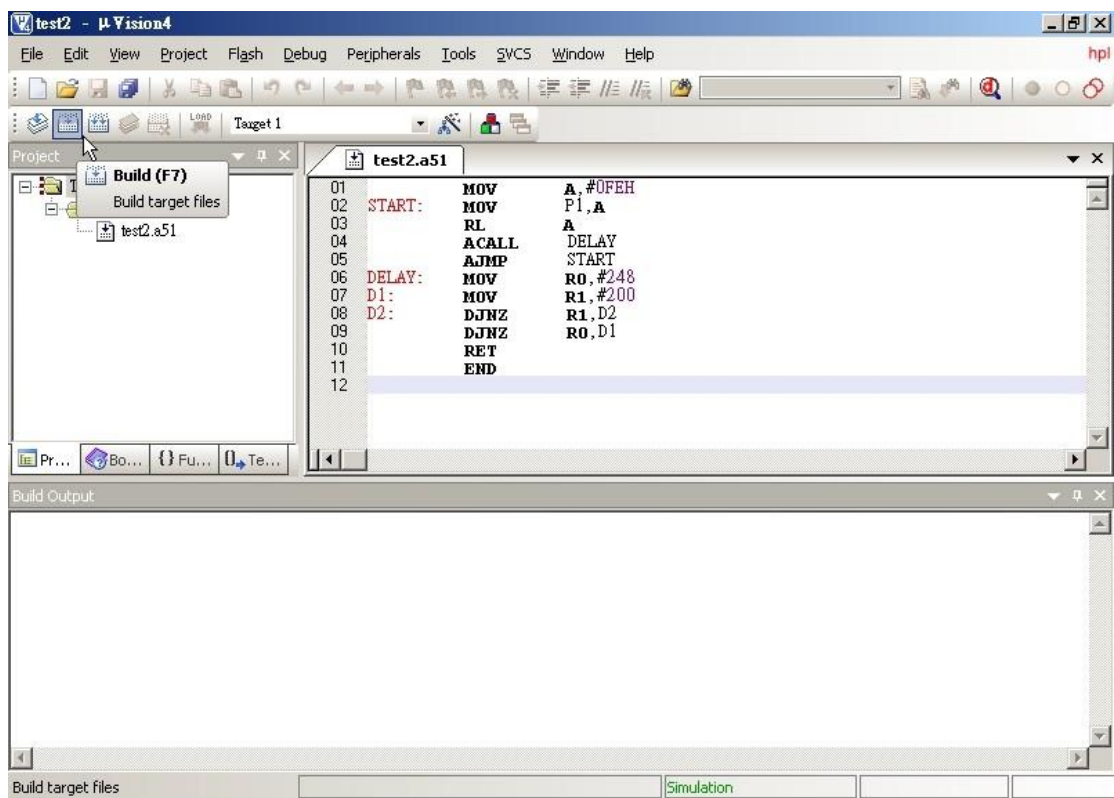


图 2-14 编译
项目

编译完成后，在下方的 Build Output 区域中，可看到编译的讯息，如图 2-15 所示。若有出现错误讯息，则再根据错误讯息，回到程序中修改，编译完全正确后，才能产生正确的烧录档 test2.hex。

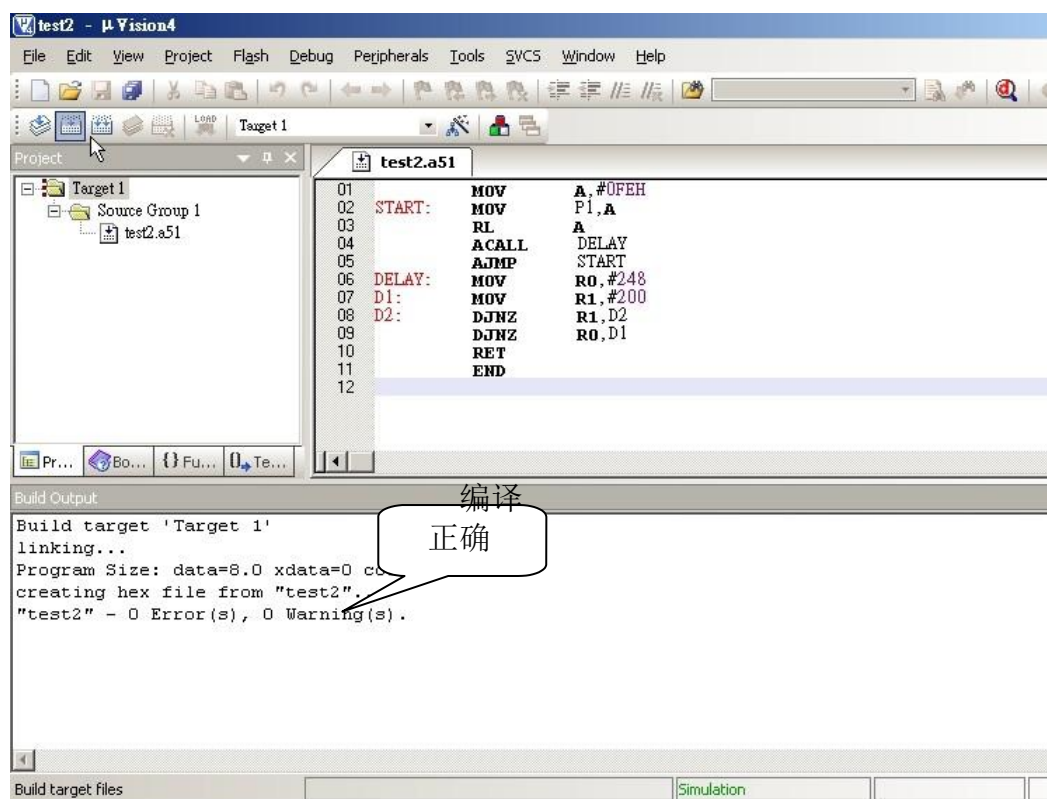










图 2-15 项目编译
正确

(8) 选取选单 Debug/Start/Stop Debug Session，或按快捷键 Ctrl+F5，或按在档案工具列右边有一个小放大镜的按钮，则进入除错(Debug)模式，并显示不同的工作窗口，如图 2-16 所示。进入除错模式之前，同样地会先出现一个小窗口，告诉你目前用的版

本是免费的评估版，有 2K ROM 大小的限制，点击确定即可进入除错模式。在除错工具列中， (Reset)按钮表示重置单芯片，并使程序回到最开头处执行。 (Run)按钮表示

执行， (Stop)按钮表示停止，当程序处于执行状态时，停止按钮才有效。 (Step Into)

按钮表示单步执行会进入函数内， (Step Over)按钮表示单步执行不会进入函数内，

 (Step Out)按钮表示离开函数， (Run to Cursor)按钮表示执行到光标所在处。

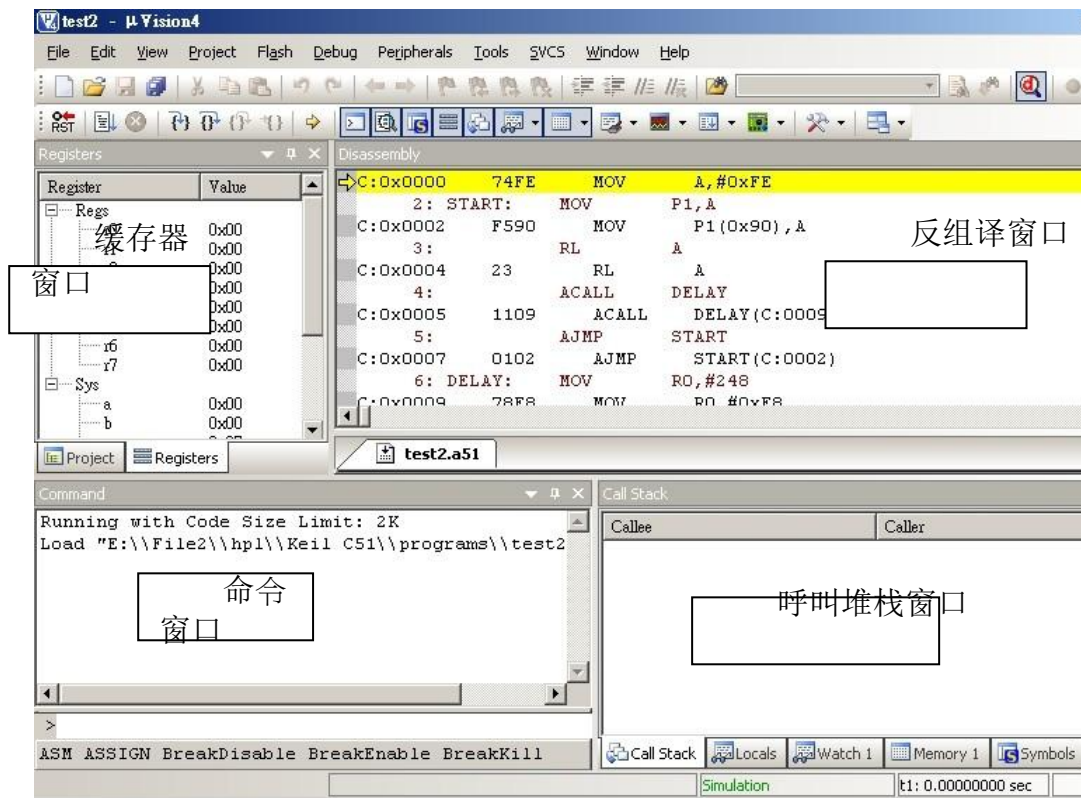


图 2-16 程序除错画面

为了要检视输出结果是否正确，则必须叫出 P1 输出埠观察输出结果。选取

Peripheral/I/O-Ports/Port 1，如图 2-17 所示。

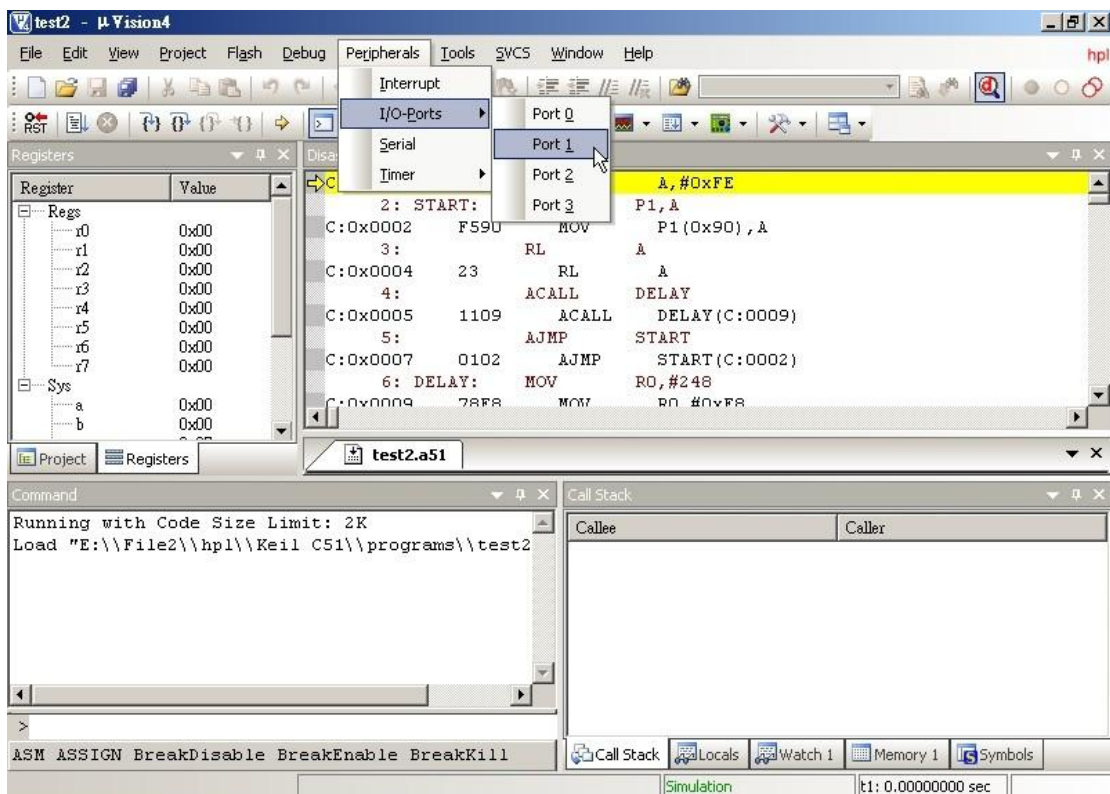


图 2-17 选取 Port 1

出现 Parallel Port 1 小窗口，并显示每一个位的值，也可移动到其他位置观察，如图 2-18 所示。

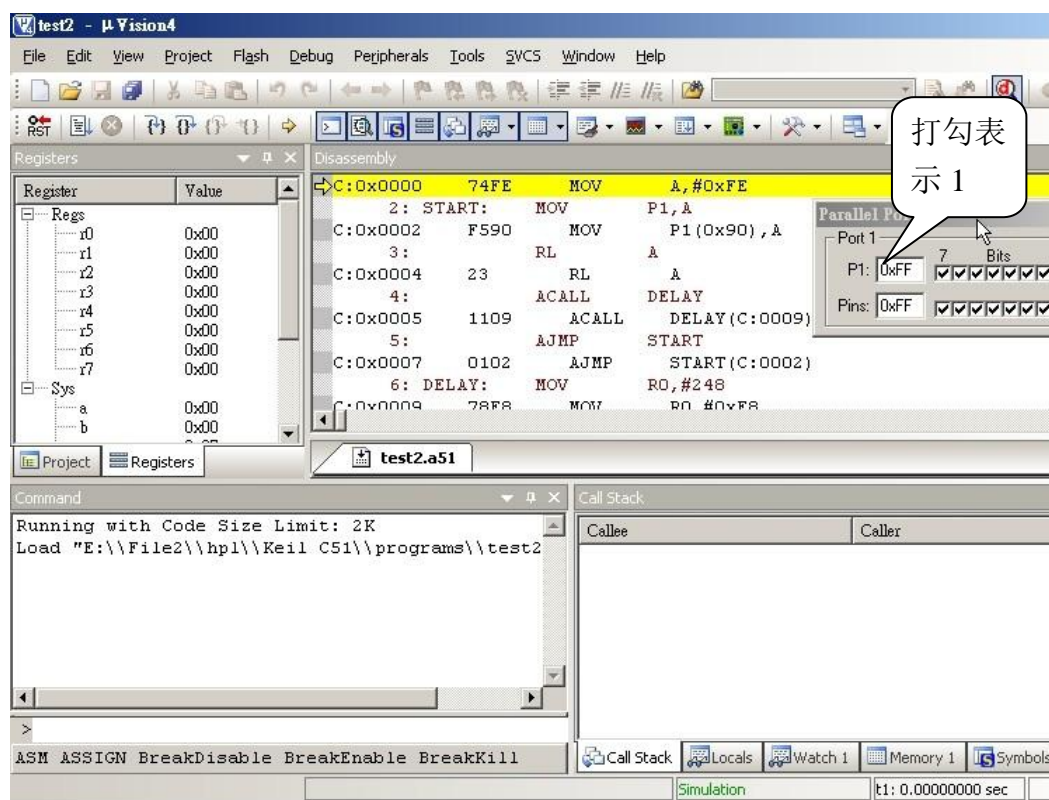


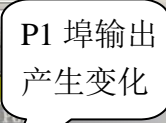
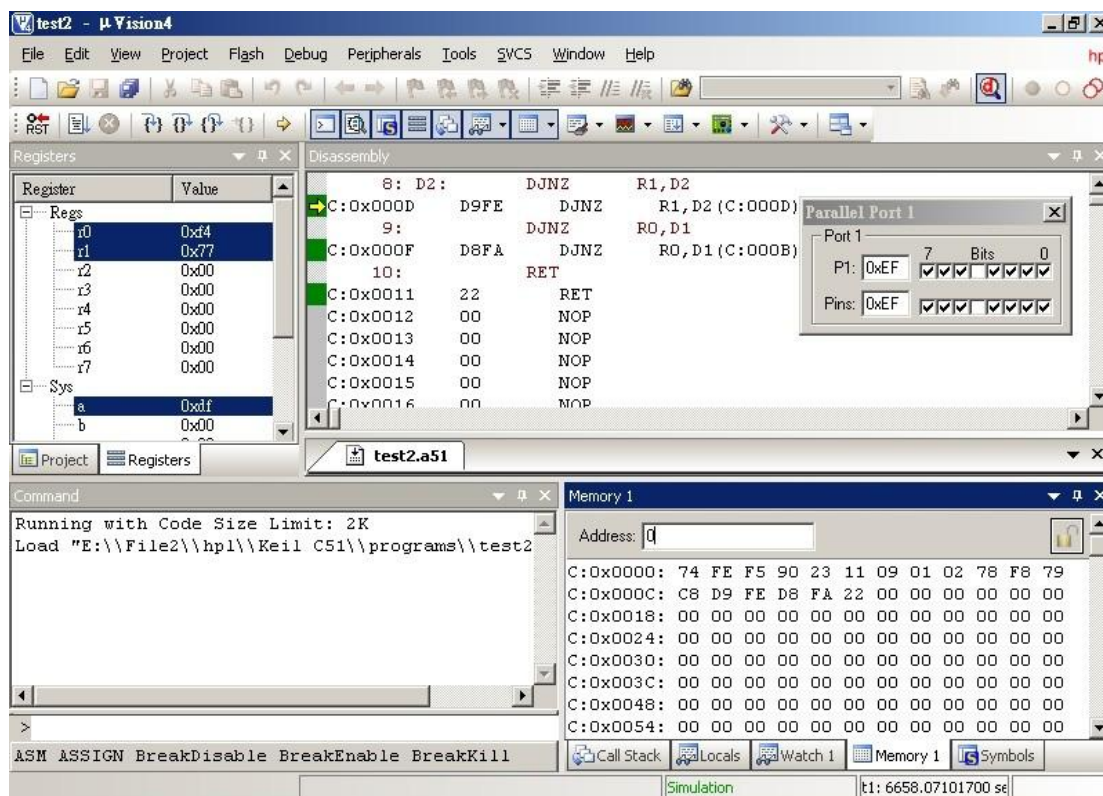


图 2-18 显示 Port 1 的窗口 最后要准备执行此程序了，先单击重置  按钮，让单芯片及程序回到最初状态，

再按下执行  按钮后，则程序开始执行。我们可以看到 Parallel Port 1 窗口中的 P1_0 位 元到 P1_7 位不断的被设定与清除，如图 2-19 所示。



View/Memory Windows/Memory 1，或按右下方的 Memory 1 按钮，然后在 Address 欄位 內輸入 0，則 Memory 1 窗口从 0x0000 开始显示运算码，如图 2-20 所示。



2-20 Memory 1

,

Project/Close Project ▼

Keil C51μVisio

Keil C51 Help

附录 1

```

STARTUP
.a51
$NOMO
D51
;-----
; This file is part of the C51 Compiler package
; Copyright (c) 1988-2005 Keil Elektronik GmbH and Keil Software, Inc.
; Version 8.01
;
; *** <<< Use Configuration Wizard in Context Menu >>> ***
;-----
;          STARTUP.A51:      This code is executed after processor reset.
;
; To translate this file use A51 with the following invocation:
;
; A51 STARTUP.A51
;
; To link the modified STARTUP.OBJ file to your application use the following
; Lx51 invocation:
;
; Lx51 your object file list, STARTUP.OBJ  controls
;
;-----
;
; User-defined <h> Power-On Initialization of Memory
;
; With the following EQU statements the initialization of memory
; at processor reset can be defined:
;
; <o> IDATALEN: IDATA memory size <0x0-0x100>
; <i> Note: The absolute start-address of IDATA memory is always 0
; <i>      The IDATA space overlaps physically the DATA and BIT areas.
IDATALEN      EQU      80H
;
; <o> XDATASTART: XDATA memory start address <0x0-0xFFFF>
; <i> The absolute start address of XDATA memory
XDATASTART    EQU      0
;
; <o> XDATALEN: XDATA memory size <0x0-0xFFFF>
; <i> The length of XDATA memory in bytes.

```

```

XDATA          ]
LEN

; <o> PDATASTART: PDATA memory start address <0x0-0xFFFF>
; <i> The absolute start address of PDATA memory
PDATASTART EQU    0H
;
; <o> PDATALEN: PDATA memory size <0x0-0xFF>
; <i> The length of PDATA memory in bytes.
PDATALEN EQU    0H
;
; </h>
;-----
;
; <h> Reentrant Stack Initialization
;
; The following EQU statements define the stack pointer for reentrant
; functions and initialized it:
;
; <h> Stack Space for reentrant functions in the SMALL model.
; <q> IBPSTACK: Enable SMALL model reentrant stack
; <i> Stack space for reentrant functions in the SMALL model.
IBPSTACK EQU    0          ; set to 1 if small reentrant is used.
; <o> IBPSTACKTOP: End address of SMALL model stack <0x0-0xFF>
; <i> Set the top of the stack to the highest location.
IBPSTACKTOP EQU    0xFF +1      ; default 0FFH+1
; </h>
;
; <h> Stack Space for reentrant functions in the LARGE model.
; <q> XBPSTACK: Enable LARGE model reentrant stack
; <i> Stack space for reentrant functions in the LARGE model.
XBPSTACK EQU    0          ; set to 1 if large reentrant is used.
; <o> XBPSTACKTOP: End address of LARGE model stack <0x0-0xFFFF>
; <i> Set the top of the stack to the highest location.
XBPSTACKTOP EQU    0xFFFF +1    ; default 0FFFFH+1
; </h>
;
; <h> Stack Space for reentrant functions in the COMPACT model.
; <q> PBPSTACK: Enable COMPACT model reentrant stack
; <i> Stack space for reentrant functions in the COMPACT model.

```



```

        PBPST          ]          ; set to 1 if compact reentrant is
ACK
        ; <o> PBPSTACKTOP: End address of COMPACT model stack <0x0-0xFFFF>
        ; <i> Set the top of the stack to the highest location.
PBPSTACKTOP      EQU      0xFF +1      ; default 0FFH+1
        ; </h>
        ; </h>
        ;-----
        ;
        ; Memory Page for Using the Compact Model with 64 KByte xdata RAM
        ; <e>Compact Model Page Definition
        ;
        ; <i> Define the XDATA page used for PDATA variables.
        ; <i> PPAGE must conform with the PPAGE set in the linker invocation.
        ;
        ; Enable pdata memory page initialization
PPAGEENABLE EQU      0          ; set to 1 if pdata object are used.
        ;
        ; <o> PPAGE number <0x0-0xFF>
        ; <i> uppermost 256-byte address of the page used for PDATA
variables. PPAGE   EQU      0
        ;
        ; <o> SFR address which supplies uppermost address byte <0x0-0xFF>
        ; <i> most 8051 variants use P2 as uppermost address byte
PPAGE_SFR      DATA      0A0H
        ;
        ; </e>
        ;-----

        ; Standard SFR
Symbols ACC      DATA
          0E0H B   DATA
          0F0H SP  DATA
81H DPL DATA    82H
DPH      DATA    83H

NAME
?C_STARTUP

```

```
?C_C51STAR      SEGM      C
?STACK          SEGMENT  IDATA
```

```
                RSEG
                ?STACK
DS              1
```

```
                EXTRN CODE
                (?C_START) PUBLIC
                ?C_STARTUP
```

```
                CSEG      AT      0
?C_STARTUP: LJMP  STARTUP1
```

```
RSEG            ?C_C51STARTUP
```

STARTUP1:

```
IF IDATALEN <> 0
    MOV R0,#IDATALEN - 1
CLR    A IDATALOOP:
    MOV    @R0,A
    DJNZ   R0,IDATALOOP
ENDIF
```

```
IF XDATALEN <> 0
    MOV DPTR,#XDATASTART
    MOV    R7,#LOW (XDATALEN)
    IF (LOW (XDATALEN)) <> 0
        MOV R6,#(HIGH (XDATALEN)) +1
    E
LSE
    MOV R6,#HIGH (XDATALEN)
```

```
    E
NDIF
    CLR    A
    XDATALOOP:
    MOVX    @DPTR,A
    INC     DPTR
    DJNZ    R7,XDATALOOP
    DJNZ    R6,XDATALOOP
ENDIF
```

2C_G51STAR SEGM C
IF PPAGEENABLE <> 0

```

MOV PPAGE_SFR,#PPAGE

E
NDIF

IF PDATALEN <> 0
    MOV R0,#LOW (PDATASTART)
    MOV R7,#LOW (PDATALEN)
    CLR A
    PDATALOOP:
    MOVX @R0,A
    INC R0
    DJNZ R7,PDATALOOP
ENDIF

```

```

IF IBPSTACK <> 0
    EXTRN DATA (?C_IBP)

    MOV ?C_IBP,#LOW IBPSTACKTOP

E
NDIF

```

```

IF XBPSTACK <> 0
    EXTRN DATA (?C_XBP)

    MOV ?C_XBP,#HIGH XBPSTACKTOP
    MOV ?C_XBP+1,#LOW XBPSTACKTOP

E
NDIF

```

```

IF PBPSTACK <> 0
    EXTRN DATA (?C_PBP)
    MOV ?C_PBP,#LOW PBPSTACKTOP
ENDIF

```

```

MOV SP,#?STACK-1

```

; This code is required if you use L51_BANK.A51 with Banking Mode 4

; <h> Code Banking

; <q> Select Bank 0 for L51_BANK.A51 Mode 4

#if 0

; <i> Initialize bank mechanism to code bank 0 when using L51_BANK.A51 with Banking Mode 4.

EXTRN CODE (?B_SWITCH0)


```
CALL    ?B_SWITCH0    ; init bank
#       mechanism to code bank 0
endif
;
</h>    LJMP

        ?C_START

END
```

附录 2

AT89X5

1.H

/*-----

Header file for the low voltage Flash Atmel AT89C51 and AT89LV51.

Copyright (c) 1988-2002 Keil Elektronik GmbH and Keil Software, Inc.

All rights reserved.

-----*/

#ifndef __AT89X51_H_

#define __AT89X51_H_____

/*-----

----- Byte Registers

-----*/

sfr P0 = 0x80;

sfr SP = 0x81;

sfr DPL = 0x82;

sfr DPH = 0x83;

sfr PCON = 0x87;

sfr TCON = 0x88;

sfr TMOD = 0x89;

sfr TL0 = 0x8A;

sfr TL1 = 0x8B;

sfr TH0 = 0x8C;

sfr TH1 = 0x8D;

sfr P1 = 0x90;

sfr SCON = 0x98;

sfr SBUF = 0x99;

sfr P2 = 0xA0;

sfr IE = 0xA8;

sfr P3 = 0xB0;

sfr IP = 0xB8;

sfr PSW =

0xD0; sfr ACC =

0xE0; sfr B =

0xF0;

/*-----

----- P0 Bit Registers

-----*/

```

        sbit P0_0 =
0x80; sbit P0_1 =
0x81; sbit P0_2 =
0x82; sbit P0_3 =
0x83; sbit P0_4 =
0x84; sbit P0_5 =
0x85; sbit P0_6 =
0x86; sbit P0_7 =
0x87;

/*-----
----- PCON Bit Values
-----
---*/
#define          0x0
IDL_          1

#define PD_      0x    /* Alternate
          02      definition */

#define          0x
GF0_          04

#define          0x

#define          0
SMOD          00

/*-----
----- TCON Bit Registers
-----
---*/
sbit IT0      =
0x88; sbit IE0 =
0x89; sbit IT1 =
0x8A; sbit IE1 =
0x8B; sbit TR0 =
0x8C; sbit TF0 =
0x8D; sbit TR1 =
0x8E; sbit TF1 =
0x8F;

/*-----
----- TMOD Bit Values
-----
---*/

```

```
    #define    T0_M0_  
0x01  
    #define    T0_M1_  
0x02
```

```

#define T0_CT_      0x04
#define T0_GATE_ 0x08
#define T1_M0_      0x10
#define T1_M1_      0x20
#define T1_CT_      0x40
#define T1_GATE_ 0x80

#define T1_MASK_ 0xF0
#define T0_MASK_ 0x0F

/*-----
----- P1 Bit Registers
-----*/

sbit P1_0 =
0x90; sbit P1_1 =
0x91; sbit P1_2 =
0x92; sbit P1_3 =
0x93; sbit P1_4 =
0x94; sbit P1_5 =
0x95; sbit P1_6 =
0x96; sbit P1_7 =
0x97;

/*-----
----- SCON Bit Registers
-----*/

sbit RI =
0x98; sbit TI =
0x99; sbit RB8
= 0x9A;
sbit TB8 = 0x9B;
sbit REN = 0x9C;
sbit SM2 = 0x9D;
sbit SM1 = 0x9E;
sbit SM0 = 0x9F;

/*-----
----- P2 Bit Registers
-----*/

sbit P2_0 = 0xA0;
sbit P2_1 = 0xA1;

```



```

        sbit P2_2 =
0xA2; sbit P2_3 =
0xA3; sbit P2_4 =
0xA4; sbit P2_5 =
0xA5; sbit P2_6 =
0xA6; sbit P2_7 =
0xA7;

/*-----
----- IE Bit Registers
-----
---*/
        sbit      =          /* 1=Enable External interrupt
        sbit      = 0xA9;      /* 1=Enable Timer 0 interrupt
        sbit      =          /* 1=Enable External interrupt
        sbit      =          /* 1=Enable Timer 1 interrupt
        sbit      =          /* 1=Enable Serial port
        sbit      =          /* 1=Enable Timer 2 interrupt

        sbit EA      = 0xAF;          /* 0=Disable all
interrupts */

/*-----
----- P3 Bit Registers (Mnemonics &
Ports)
-----
---*/
        sbit P3_0 =
0xB0; sbit P3_1 =
0xB1; sbit P3_2 =
0xB2; sbit P3_3 =
0xB3; sbit P3_4 =
0xB4; sbit P3_5 =
0xB5; sbit P3_6 =
0xB6; sbit P3_7 =
0xB7;

        sbit RXD      = 0xB0;          /* Serial
data input */ sbit TXD      = 0xB1;          /* Serial
data output */ sbit INT0 = 0xB2;          /* External
interrupt 0 */ sbit INT1 = 0xB3;          /* External
interrupt 1 */

```

```
    sbit T0    = 0xB4;        /* Timer 0 external
input */
    sbit T1    = 0xB5;        /* Timer 1 external
input */
    sbit WR    = 0xB6;        /* External data memory write
strobe */
    sbit RD    = 0xB7;        /* External data memory read
strobe */
```

```

/*-----
----- IP Bit Registers
-----*/

sbit PX0    =
0xB8; sbit PT0    =
0xB9; sbit PX1    =
0xBA; sbit PT1    =
0xBB; sbit PS     =
0xBC; sbit PT2    =
0xBD;

/*-----
----- PSW Bit Registers
-----*/

sbit P= 0xD0;
sbit F1    = 0xD1;
sbit OV     = 0xD2;
sbit RS0    = 0xD3;
sbit RS1    = 0xD4;
sbit F0     = 0xD5;
sbit AC     = 0xD6;
sbit CY     = 0xD7;

/*-----
----- Interrupt Vectors:
Interrupt Address = (Number * 8) + 3
-----*/

#define          /* 0x03 External Interrupt
-----
#define          /* 0x0B Timer 0 */
-----
#define          /* 0x13 External Interrupt
-----
#define          /* 0x1B Timer 1 */
-----
#define          /* 0x23 Serial port */
SIG_VECTOR

#endif

```