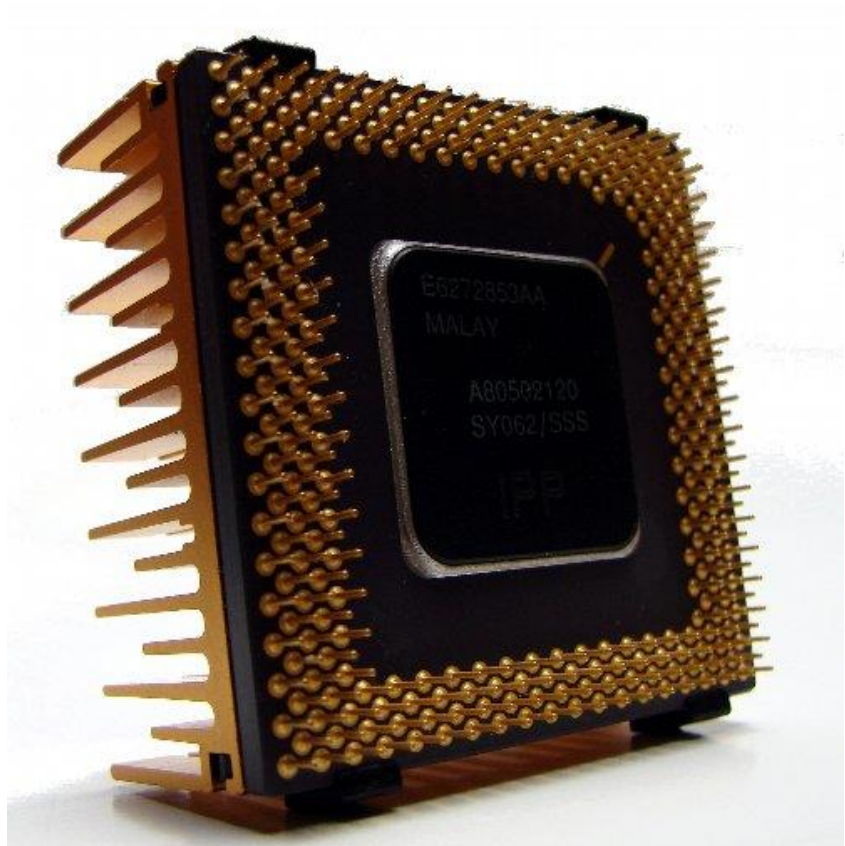


PROYECTO

MATERIAL INDIVIDUAL F-II



2020

Proyecto de evaluación del rendimiento

F-II. Implementación de un benchmark reducido y evaluación del procesamiento de arquitecturas PC convencionales

Arquitectura de los Computadores

Grado en Ingeniería Informática

Dpto. Tecnología Informática y Computación

Universidad de Alicante

Material individual de la Fase II

INTRODUCCIÓN

I. CONTENIDOS, ENTREGA Y EVALUACIÓN INDIVIDUAL

El material adicional para el desarrollo de la Fase II del proyecto consta de tres partes:

- PARTE 1: Información sobre la puesta a punto de programas de ensamblador inline con C++ empleando la herramienta de desarrollo de software Visual Studio .NET. Ejercicios que deberán ser resueltos de manera individual. Los ejercicios forman parte de la guía de puesta a punto. Además, se propone el desarrollo de un programa en ensamblador inline en el apartado VII.
 - **Se aconseja que el estudiante realice esta parte antes de abordar la primera parte grupal de esta fase.**
- PARTE 2: Información para SSE
 - **Se aconseja que el estudiante realice esta parte antes de abordar la segunda parte grupal de esta fase.**
- PARTE 3: Información del benchmark SPEC para el ejercicio individual.
 - **Se aconseja que el estudiante realice esta parte antes de abordar la tercera parte grupal de esta fase.**

Evaluación Individual: Esta parte se evaluará mediante un examen tipo test en la plataforma Moodle, donde se preguntará sobre aspectos de desarrollo, implementación y sobre las plataformas usadas durante estos ejercicios individuales.

Material individual: PARTE 1 / 3

USO DE ENSAMBLADOR EMBEBIDO

II. VISUAL STUDIO .NET

Visual Studio .NET implementa dos contenedores conceptuales que facilitan la administración de los elementos necesarios para el desarrollo de software, como pueden ser: referencias, conexiones de datos, carpetas y archivos. Estos contenedores se denominan soluciones y proyectos. Una solución incluye uno o varios proyectos más los archivos y metadatos que ayudan a definir la solución como un todo. Las carpetas de soluciones permiten organizar proyectos relacionados en grupos y gestionar esos grupos de proyectos. Un proyecto incluye un conjunto de archivos de código fuente más los metadatos relacionados, como referencias de componentes e instrucciones de generación. Normalmente, los proyectos crean uno o más archivos de salida cuando se generan.

Cuando se crea un nuevo proyecto, Visual Studio genera una solución automáticamente, pudiendo agregar otros proyectos a la solución. El Explorador de soluciones proporciona una vista gráfica de toda la solución que ayuda a administrar los proyectos y archivos durante el desarrollo de la aplicación. También pueden crearse soluciones en blanco, es decir, sin proyectos, lo que permite modificar los proyectos, las carpetas y los archivos que la conforman de forma independiente. Dado que cada proyecto o solución comprende un directorio y su contenido, las soluciones y los proyectos se pueden mover, copiar o eliminar empleando el Explorador de Windows.

III. CREAR UNA NUEVA SOLUCIÓN Y UN NUEVO PROYECTO DE C++

En este primer ejercicio guiado se creará una nueva Solución Visual Studio .NET, denominada *EjerciciosEnsamblador*, que se utilizará para desarrollar los ejercicios de programación contenidos en esta práctica, y un nuevo Proyecto de Visual C++, denominado *Ejemplo1*. Para ello, realizar las siguientes acciones:

1. Abrir Visual Studio .NET.
2. Acceder al menú **Archivo** de Visual Studio .NET, expandir la opción **Nuevo** y seleccionar la opción **Proyecto...**
3. En el cuadro de diálogo **Nuevo Proyecto**, realizar las siguientes acciones:
 - a. En la sección **Plantillas instaladas**, expandir la opción **Visual C++**, elegir la opción **General** y seleccionar **Proyecto vacío**.
 - b. Comprobar que la casilla de verificación **Crear directorio para la solución** se encuentra activada.
 - c. En el cuadro de texto **Nombre** introducir *Ejemplo1*.
 - d. Seleccionar una carpeta como ubicación para la solución de Visual Studio mediante el botón **Examinar...**
 - e. En el cuadro de texto **Nombre de la solución** se mostrará el mismo nombre asignado al Proyecto de Visual C++. Modificar su contenido, introduciendo *EjerciciosEnsamblador* en el cuadro de texto **Nombre de la solución**.
 - f. Finalmente, comprobar que está activada la casilla de verificación **Crear directorio para la solución** y hacer clic en **Aceptar**.

Utilizando el Explorador de archivos de Windows, se puede comprobar que se ha creado la carpeta *EjerciciosEnsamblador* en la ubicación deseada. Y que en su contenido se encuentra el archivo de solución *EjerciciosEnsamblador.sln* y la carpeta *Ejemplo1*. A su vez, la carpeta *Ejemplo1* contiene los archivos que son propios de un Proyecto de Visual C++.

Una vez cerrado Visual Studio .NET, para acceder a la Solución *EjerciciosEnsamblador* y al proyecto *Ejemplo1*, bastará con hacer doble clic sobre el archivo *EjerciciosEnsamblador.sln* desde su ubicación, empleando para ello el Explorador de archivos de Windows.

IV. CREAR UN ARCHIVO DE CÓDIGO

Una vez creada la solución *EjerciciosEnsamblador* y el Proyecto de Visual C++ *Ejemplo1*, se creará un archivo de código denominado *Suma.cpp*. Para ello, hacer:

1. Sobre la ventana **Explorador de soluciones**, hacer clic con el botón derecho sobre el nombre del Proyecto de Visual C++ *Ejemplo1*, expandir la opción **Agregar** y seleccionar la opción **Nuevo elemento...**
2. En el cuadro de diálogo **Agregar nuevo elemento**, realizar las siguientes acciones:
 - a. En la sección **Plantillas instaladas**, expandir la opción **Visual C++**, elegir la opción **Código** y seleccionar **Archivo C++ (.cpp)**.
 - b. En el cuadro de texto **Nombre** introducir *Suma*.
 - c. Comprobar que el archivo se creará en la ubicación adecuada.
 - d. Hacer clic en **Agregar**.

Empleando el **Explorador de soluciones** de Visual Studio .NET y el **Explorador de archivos** de Windows, se puede comprobar que el archivo *Suma.cpp* ha sido creado.

3. Agregar el siguiente código sobre el archivo de código *Suma.cpp*.

```
// Ensamblador en línea. Procesador: x86 (32 bits)

#include <stdio.h>

int suma(int a, int b);

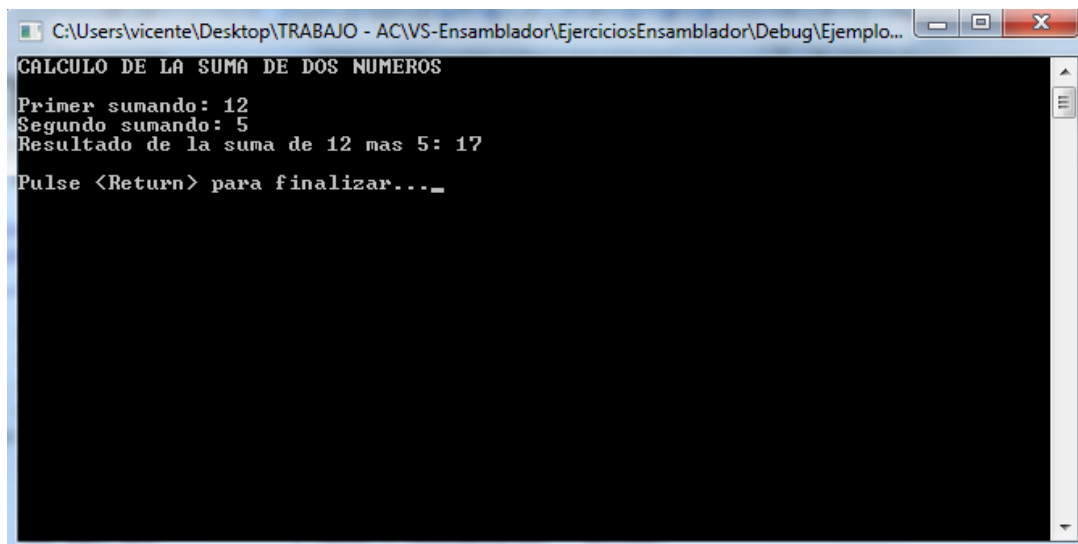
int main( void )
{
    int x = 0;
    int y = 0;

    printf( "CALCULO DE LA SUMA DE DOS NUMEROS\n");
    printf_s("\n");
    printf_s("Primer sumando: ");
    scanf_s("%d",&x);
    printf_s("Segundo sumando: ");
    scanf_s("%d",&y);
    getchar();
    printf_s("Resultado de la suma de %d mas %d: %d\n", x, y, suma(x, y));
```

```
printf_s("\n");
printf_s("Pulse <Return> para finalizar...");
getchar();
}

int suma(int a, int b)
{
    __asm
    {
        mov eax, a      ; Almacena en EAX el primer argumento
        mov ecx, b      ; Almacena en ECX el segundo argumento
        add eax, ecx     ; Suma de los operandos, deja el resultado en eax
        // Devuelve el resultado de EAX
    }
}
```

4. Una vez editado el código, seleccionar la opción **Guardar Suma.cpp** del menú **Archivo**, o, hacer clic en el botón **Guardar** de la **Barra de herramientas** de Visual Studio .NET.
5. Seleccionar la opción **Generar solución** del menú **Generar**. Si se produjeran errores durante el proceso de compilación deberán resolverse antes de ejecutar el código.
6. Para comprobar el funcionamiento del programa, seleccionar la opción **Iniciar la depuración** del menú **Depurar**, o bien, hacer clic sobre el botón **Iniciar depuración** de la **Barra de herramientas** de Visual Studio.



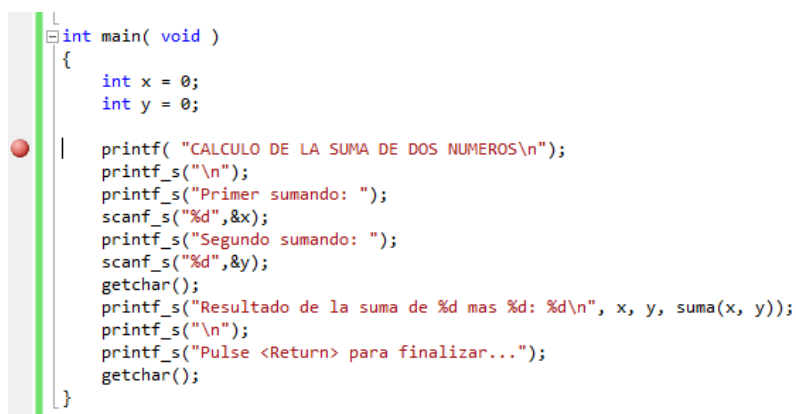
V. DEPURACIÓN PASO A PASO

El código de los programas informáticos puede contener diversos tipos de errores. La forma de detectar los errores lógicos es probar el programa para comprobar que el resultado es el esperado. Las pruebas son una parte integrante del proceso de desarrollo de software. En algunos casos de error, las pruebas pueden indicar que el resultado de un programa es incorrecto, aunque pueden no proporcionar ninguna pista acerca de qué parte del código ha causado el problema en la ejecución. En estos casos es útil emplear la depuración paso a paso.

La depuración paso a paso consiste en ejecutar línea a línea el código para inspeccionar los valores que van adoptando las variables a lo largo del flujo de la ejecución del código. El menú **Depurar** contiene dos comandos para avanzar paso a paso por el código: **Paso a paso por instrucciones** y **Paso a paso por procedimientos**. Es posible iniciar estos métodos de depuración paso a paso pulsando las teclas <F11> y <F10>, respectivamente.

Los métodos de depuración **Paso a paso por instrucciones** y **Paso a paso por procedimientos** sólo se diferencian en la forma en que tratan las llamadas a funciones. Ambos comandos indican al depurador que ejecute la siguiente línea de código. Si la línea contiene una llamada a una función, **Paso a paso por instrucciones** sólo ejecuta la llamada en sí y, a continuación, se detiene en la primera línea de código incluida en la función. Sin embargo, **Paso a paso por procedimientos** ejecuta toda la función y después se detiene en la primera línea que está fuera de ella.

En algunas ocasiones, durante la depuración, es posible que se desee ejecutar el código hasta un punto determinado y después interrumpir la ejecución. En estos casos, se introducirá un Punto de interrupción en ese lugar, pudiendo continuar la ejecución en el modo de depuración paso a paso. Para introducir un Punto de interrupción, basta con seleccionar la línea de código deseada haciendo clic a la izquierda de la línea sobre la barra vertical de selección de líneas de código. Los puntos de interrupción se representan mediante un círculo de color rojo. Para eliminar un Punto de interrupción basta con hacer clic sobre él.



```

1 int main( void )
2 {
3     int x = 0;
4     int y = 0;
5
6     printf( "CALCULO DE LA SUMA DE DOS NUMEROS\n");
7     printf_s("\n");
8     printf_s("Primer sumando: ");
9     scanf_s("%d",&x);
10    printf_s("Segundo sumando: ");
11    scanf_s("%d",&y);
12    getchar();
13    printf_s("Resultado de la suma de %d mas %d: %d\n", x, y, suma(x, y));
14    printf_s("\n");
15    printf_s("Pulse <Return> para finalizar...");
16    getchar();
17 }
  
```

En cualquier momento puede detenerse la depuración seleccionando la opción **Detener depuración** del menú **Depurar**, o bien, hacer clic sobre el botón **Detener depuración** de la **Barra de herramientas** de Visual Studio.

Como ejercicio, se propone que se realice la depuración del archivo de código *Suma.cpp*, introduciendo un punto de interrupción al principio del programa y ejecutando paso a paso por procedimientos hasta la línea de código que incluye el punto de llamada a la función *suma*.

VI. OPCIONES DE COMPILACIÓN

Para acceder a las opciones de compilación, seleccionar **Propiedades de Ejemplo2...** del menú **Proyecto** de Visual Studio .NET. En el cuadro de diálogo **Páginas de propiedades de Ejemplo2**, expandir la opción **Propiedades de Configuración**. Finalmente, al expandir la opción **C/C++** aparecen las opciones de compilación agrupadas por categorías. Recorrer cada una de las categorías para conocer sus opciones. Algunas de las opciones más relevantes son:

- Optimización. Deberá quedar **Deshabilitada (/Od)** al desarrollar programas de prueba de rendimiento.
- Archivos de salida. Para obtener el archivo de ensamblador, seleccionar **Ensamblado con código fuente (/FAs)** en la opción **Resultados de Ensamblado**. Al generar, se crea el archivo .asm en la carpeta *Debug* de la carpeta del Proyecto de Visual C++. Así, al ajustar esta opción y generar nuevamente la solución *Ejemplo2*, puede comprobarse que en la carpeta *Debug* de la carpeta del proyecto se habrá creado el archivo *Ejemplo2.asm*. Este archivo contiene el código ensamblador correspondiente al programa. Conviene comprobar cómo cada instrucción de ensamblador incluida en el bloque `__asm{}` corresponde con una única instrucción después de la compilación.

Línea de comandos. Se visualiza las opciones de la línea de comandos del compilador.

VII. EJERCICIO

Se propone desarrollar en lenguaje ensamblador del x86 un programa ensamblador que realice la ordenación de un vector mediante el algoritmo de la burbuja.

A continuación se expresa el algoritmo de la burbuja para ordenación de vectores:

```
para i=0 hasta n-1 hacer
  para j=n hasta i+1 hacer
    si (vector[j-1]<vector[j])
      intercambiar(vector[j-1],vector[j])
    fin_si
  fin_para
fin_para
```

Realizar el programa inicializando el vector al declararlo en C++ con una longitud de 20 componentes. La lógica del algoritmo de la burbuja debe ser expresamente implementada en ensamblador.

ANEXO. TIPOS DE DATOS EN C/C++

La ocupación en memoria y el rango de valores de los tipos de datos más comunes en C++ se muestran en la siguiente tabla.

Tipo de Datos	Ocupación en memoria	Rango
unsigned char	8 bits	0 a 255
char	8 bits	-128 a 127
short int	16 bits	-32,768 a 32,767
unsigned int	32 bits	0 a 4,294,967,295
int	32 bits	-2,147,483,648 a 2,147,483,647
unsigned long	32 bits	0 a 4,294,967,295
enum	16 bits	-2,147,483,648 a 2,147,483,647
long	32 bits	-2,147,483,648 a 2,147,483,647
float	32 bits	3.4×10^{-38} a $3.4 \times 10^{+38}$ (6 decimales)
double	64 bits	1.7×10^{-308} a $1.7 \times 10^{+308}$ (15 decimales)
long double	80 bits	3.4×10^{-4932} a $1.1 \times 10^{+4932}$

Material individual: PARTE 2/3

USO DE EXTENSIONES SIMD

VIII. CONTENIDOS

Esta parte tiene el objetivo de comparar dos tipos de arquitecturas, en concreto SISD y SIMD, según la taxonomía de Flynn y conocer con detalle el repertorio de instrucciones de una arquitectura CISC. **Se aconseja realizar esta parte individual antes de abordar la segunda parte grupal de esta fase.**

El objetivo de esta parte es que individualmente el alumno conozca las capacidades de paralelización de cálculo que incorporan los procesadores de propósito general. Esto es, comprender cómo se realiza la programación con las extensiones SIMD de las arquitecturas superescalares de Intel/AMD. Para ello, se utilizará MICROSOFT VISUAL STUDIO .NET C++ en entorno WINDOWS.

IX. DOCUMENTACIÓN EXTENSIONES SIMD (MMX, SSE)

Para la documentación referente a las extensiones SIMD se utilizará la página web a la que se hace referencia abajo como guía de documentación. El enlace contiene información referente a programación con tecnología MMX y SSE, descripción de instrucciones, modelo de trabajo, ejemplos, etc.

<http://www.tommesani.com/Docs.html>

Adicionalmente, es muy interesante la consulta de documento de Intel 'Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 1'. En el capítulo 5 se incluye una descripción del juego de instrucciones de IA-32 incluido el juego de instrucciones MMX, SSE, SSE2 y SSE3. Además, los capítulos 9 y 10 proporcionan información sobre el desarrollo de programas empleando instrucciones MMX y SSE, respectivamente.

<http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-1-manual.html>

X. EJERCICIOS INDIVIDUALES

En esta parte se pretende testear varios algoritmos utilizando las tecnologías MMX/SSE para ello se propone compilar y ejecutar los siguientes proyectos, y analizar los resultados obtenidos. El código fuente de estos proyectos se incluye como material adicional de esta parte.

NET Bubble Sort SEE (comparando assembler + C)

Las cuestiones a resolver son:

- ¿Qué hace el programa?
- ¿Se utilizan extensiones MMX o SSE?
- Comenta el funcionamiento de la función: `void sortDoublesSSE(Int32 byteCount, double* values)`
- ¿Qué ganancia obtenemos con el algoritmo MMX/SSE con respecto al algoritmo secuencial? Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.

NET Data Transfer (comparando assembler + C)

Las cuestiones a resolver son:

- ¿Qué hace el programa?
- Explica las siguientes instrucciones: shufpd, cmpltpd, movmskpd
- Explica el funcionamiento de la siguiente función: `int DataTransferOptimised(int* piDst, int* piSrc, unsigned long SizeInBytes)`
- ¿Qué ganancia obtenemos con el algoritmo optimizado mediante extensiones SIMD con respecto al algoritmo secuencial? Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.

NET Inner Product (comparando assembler + C)

Las cuestiones a resolver son:

- ¿Qué hace el programa?
- Comenta la siguiente función: `float sse3_inner(const float* a, const float* b, unsigned int size)`
- ¿Qué ganancia obtenemos con el algoritmo optimizado mediante extensiones SIMD con respecto al algoritmo secuencial? Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.

NET MatrixVectorMult (comparando assembler + C)

Las cuestiones a resolver son:

- ¿Qué hace el programa?
- ¿Qué ganancia obtenemos con el algoritmo optimizado mediante extensiones SIMD con respecto al algoritmo secuencial? Realiza una batería de pruebas y muéstralo utilizando gráficas explicativas.

Material individual: PARTE 3/3

USO DEL PAQUETE DE EVALUACIÓN SPEC

XI. INTRODUCCIÓN A SPEC

Standard Performance Evaluation Corporation (SPEC), es un consorcio sin ánimo de lucro que incluye a vendedores, integradores de sistemas, universidades, grupos de investigación, publicadores y consultores de todo el mundo. Tiene dos objetivos: crear un benchmark estándar para medir el rendimiento de computadores y controlar y publicar los resultados de estos tests.

SPEC CPU2000

SPEC CPU2000 es un benchmark producido por la SPEC. Fue creado con el fin de proveer una medida de rendimiento que pudiese ser usada para comparar cargas de trabajo intensivas en cómputo en distintos computadores. Contiene dos benchmark suites: **CINT2000** para medir y comparar el rendimiento de computación intensiva de enteros, y **CFP2000** para medir y compara el rendimiento de computación intensiva en flotantes.

La "C" en CINT2000 y CFP2000 denotan que son benchmarks a nivel de componentes, en oposición a benchmarks de todo el sistema. Al ser intensivos en cálculos, miden el rendimiento del procesador del computador, la arquitectura de la memoria y el compilador. El CINT2000 y el CFP2000 no fuerzan la entrada/salida (unidades de disco), trabajo en red o gráficos.

Contenido

SPEC provee lo siguiente en el paquete CPU2000: las herramientas del CPU2000 para compilar, ejecutar y validar los benchmarks en una variedad de sistemas operativos, el código fuente de las herramientas, de manera que puedan ser compiladas para los sistemas no cubiertos por las herramientas pre-compiladas, el código fuente de los benchmarks, herramientas para la generación de informes de rendimiento, reglas de ejecución e informes que definen cómo deberían ser usados los benchmarks para producir resultados estándar la documentación. SPEC CPU2000 incluye herramientas para la mayoría de los sistemas operativos Unix y Windows.

CINT2000 contiene 11 aplicaciones escritas en C y una en C++ (252.eon) que son usadas como benchmarks.

- 164.gzip: Utilidad de compresión de datos.
- 175.vpr: Direccionamiento y ubicación de circuitos FPGA.
- 176.gcc: Compilador C.
- 181.mcf: Costo mínimo de flujo de red.
- 186.crafty: Programa de ajedrez.
- 197.parser: Procesamiento de lenguaje natural.
- 252.eon: Efectos producidos por distintas fuentes de luz.

- 253.perlbmk: Perl.
- 254.gap: Teoría de grupo computacional.
- 255.vortex: Base de datos orientada a objetos.
- 256.bzip2: Utilidad de compresión de datos.
- 300.twolf: Simulador de ubicación y ruteo.

CFP2000 contiene 14 aplicaciones (seis en Fortran77, cuatro en FORTRAN90 y cuatro en C) que son usadas como benchmarks:

- 168.wupwise: Cromodinámica de cuantos.
- 171.swim: Modelado de aguas poco profundas.
- 172.mgrid: Multi-grilla en campos potenciales 3D.
- 173.applu: Ecuaciones diferenciales parciales parabólicas/elípticas.
- 177.mesa: Biblioteca de gráficos 3D.
- 178.galgel: Dinámica de fluidos: análisis de inestabilidad oscilatoria.
- 179.art: Simulación de red neuronal: teoría de la resonancia adaptativa.
- 183.equake: Simulación de elementos finitos: modelado de terremotos.
- 187.facerec: Reconocimientos de imágenes: reconocimiento de rostros.
- 188.amp: Química computacional.
- 189.lucas: Teoría de los números: prueba de primalidad.
- 191.fma3d: Simulación de elementos finitos en choque.
- 200.sixtrack: Modelo de acelerador de partículas.
- 301.apsi: Problemas de temperatura, viento y distribución de contaminantes.

Resultados

El SPEC2000 provee cuatro medidas para cada componente (consistente en la media geométrica de los tiempos de ejecución de los programas). Esto es porque tiene dos clasificaciones. La primera es en cuanto a la optimización del compilador por el usuario:

- base ("conservadora"): sirve para encuadrar a los usuarios que compilan con las opciones de optimización generales ofrecidas por el compilador, de manera que tiene una serie de referencias establecidas para usar las opciones del compilador (por ej. deben usarse las mismas opciones en el mismo orden en todos los benchmarks de un mismo lenguaje, no más de cuatro opciones de optimización, y no deben usarse banderas de reivindicación).
- no base ("agresiva"): intenta encuadrar a los usuarios que intentan lograr el mejor rendimiento de sus programas. Son opcionales y tienen requerimientos mucho menos estrictos (por ej. pueden usarse diferentes opciones de compilación para distintos programas escritos en el mismo lenguaje)

La segunda, se refiere a la cantidad de tareas que se ejecutan al mismo tiempo:

- no rate (monotarea): corresponde a la velocidad de ejecución de una sola tarea.
- rate (multitarea): corresponde a la capacidad de la máquina de llevar a cabo un cierto número de tareas similares. Tradicionalmente usado para medir el rendimiento de multiprocesadores.

La combinación de estas dos clasificaciones nos otorga 4 medidas distintas para el CINT2000 y 4 para el CFP2000. SPEC usa una Sun Ultra5_10 con un procesador de 300 MHz como máquina de referencia.

XII. PREPARACIÓN DEL ENTORNO SPEC CPU 2000

Copiar el directorio "speccpu2000" en la máquina que vayas a analizar e instala SPEC 2000. Edita el fichero shrc.bat y haz las modificaciones necesarias para que SPEC encuentre la ruta del compilador que usaremos, en nuestro caso será el Visual Studio.

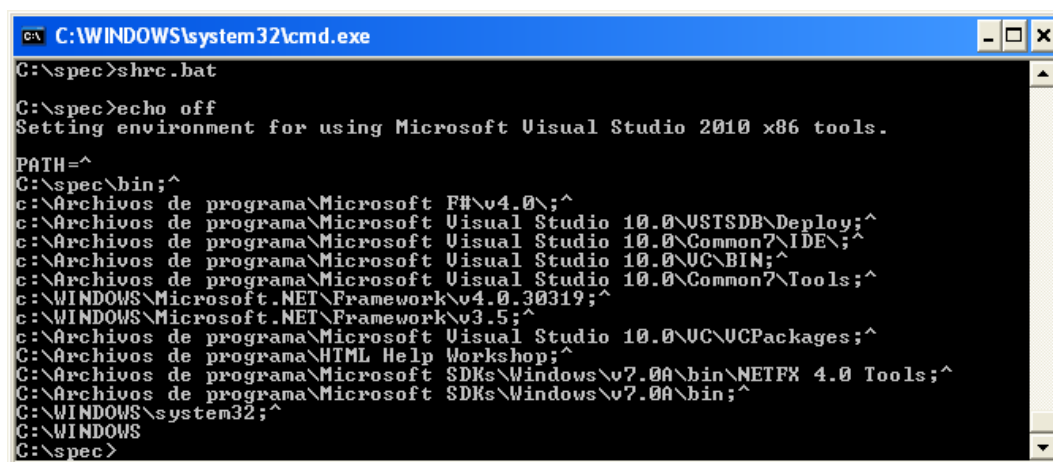
Se recomienda al alumno leer los comentarios del archivo y localizar las líneas que hay que modificar, que concretamente son las siguientes:

```
set SHRC_COMPILER_PATH_SET=yes
```

```
call "C:\Archivos de programa\Microsoft Visual Studio 11.0\VC\bin\vcvars32.bat"
```

NOTA: esta ruta puede cambiar en función de la versión de VS, se recomienda buscar previamente el archivo vcvars32.bat y poner su ruta

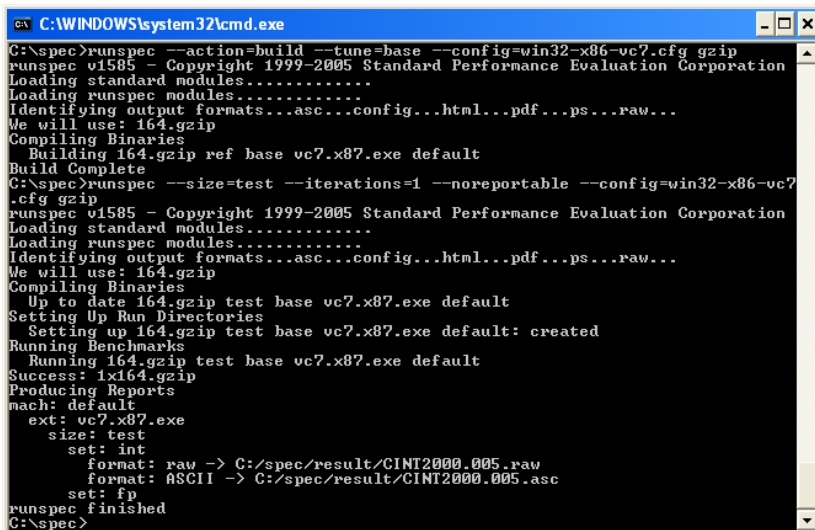
El archivo vcvars32.bat establece las variables de entorno apropiadas para habilitar generaciones desde la línea de comandos de 32 bits. El siguiente paso es ejecutar el archivo shrc.bat para establecer las variables de entorno. El resultado debe ser el siguiente:



```
C:\WINDOWS\system32\cmd.exe
C:\spec>shrc.bat
C:\spec>echo off
Setting environment for using Microsoft Visual Studio 2010 x86 tools.
PATH=^
C:\spec\bin;^
c:\Archivos de programa\Microsoft F#\v4.0\;^
c:\Archivos de programa\Microsoft Visual Studio 10.0\USTSDB\Deploy;^
c:\Archivos de programa\Microsoft Visual Studio 10.0\Common7\IDE\;^
c:\Archivos de programa\Microsoft Visual Studio 10.0\VC\BIN;^
c:\Archivos de programa\Microsoft Visual Studio 10.0\Common7\Tools;^
c:\WINDOWS\Microsoft.NET\Framework\v4.0.30319;^
c:\WINDOWS\Microsoft.NET\Framework\v3.5;^
c:\Archivos de programa\Microsoft Visual Studio 10.0\VC\UCPackages;^
C:\Archivos de programa\HTML Help Workshop;^
C:\Archivos de programa\Microsoft SDKs\Windows\v7.0A\bin\NETFX 4.0 Tools;^
C:\Archivos de programa\Microsoft SDKs\Windows\v7.0A\bin;^
C:\WINDOWS\system32;^
C:\WINDOWS
C:\spec>
```

Prueba que puedes compilar un benchmark utilizando la configuración por defecto y ejecútalo:

```
runspec --action=build --tune=base --config=win32-x86-vc7-cfg gzip
```



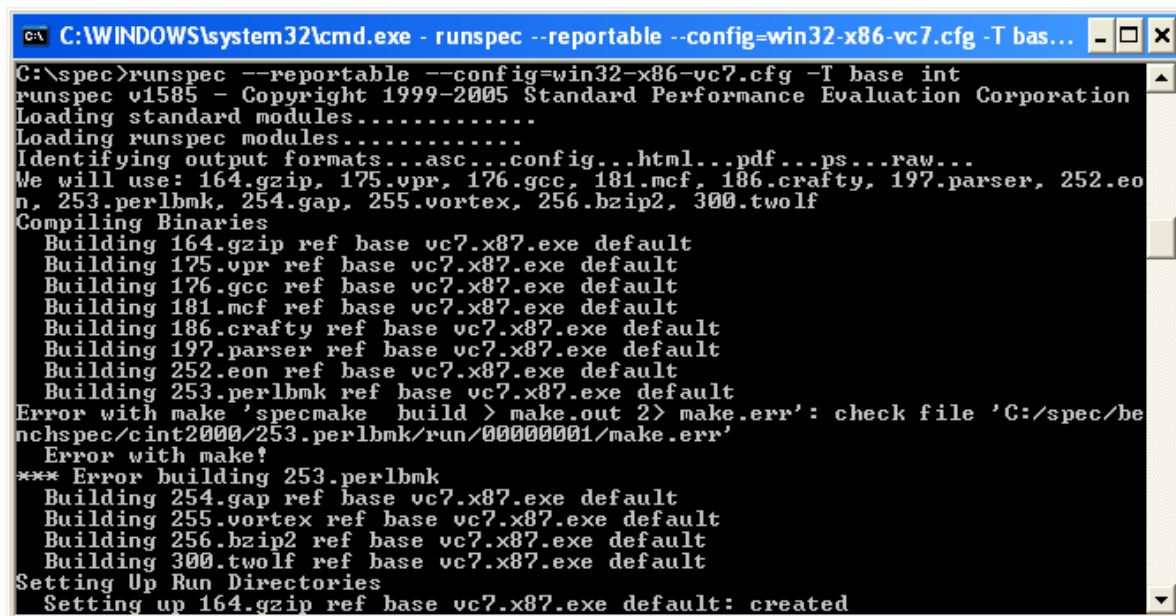
```

C:\WINDOWS\system32\cmd.exe
C:\spec>runspec --action=build --tune=base --config=win32-x86-vc7.cfg gzip
runspec v1585 - Copyright 1999-2005 Standard Performance Evaluation Corporation
Loading standard modules.....
Loading runspec modules.....
Identifying output formats...asc...config...html...pdf...ps...raw...
We will use: 164.gzip
Compiling Binaries
  Building 164.gzip ref base vc7.x87.exe default
Build Complete
C:\spec>runspec --size=test --iterations=1 --noreportable --config=win32-x86-vc7
.cfg gzip
runspec v1585 - Copyright 1999-2005 Standard Performance Evaluation Corporation
Loading standard modules.....
Loading runspec modules.....
Identifying output formats...asc...config...html...pdf...ps...raw...
We will use: 164.gzip
Compiling Binaries
  Up to date 164.gzip test base vc7.x87.exe default
Setting Up Run Directories
  Setting up 164.gzip test base vc7.x87.exe default: created
Running Benchmarks
  Running 164.gzip test base vc7.x87.exe default
Success: 1x164.gzip
Producing Reports
mach: default
  ext: vc7.x87.exe
  size: test
  set: int
    format: raw -> C:/spec/result/CINT2000.005.raw
    format: ASCII -> C:/spec/result/CINT2000.005.asc
  set: fp
runspec finished
C:\spec>

```

Ejecuta los benchmark de SPEC con el siguiente comando:

```
runspec --reportable --config=win32-x86-vc7.cfg -T base int
```



```

C:\WINDOWS\system32\cmd.exe - runspec --reportable --config=win32-x86-vc7.cfg -T bas...
C:\spec>runspec --reportable --config=win32-x86-vc7.cfg -T base int
runspec v1585 - Copyright 1999-2005 Standard Performance Evaluation Corporation
Loading standard modules.....
Loading runspec modules.....
Identifying output formats...asc...config...html...pdf...ps...raw...
We will use: 164.gzip, 175.vpr, 176.gcc, 181.mcf, 186.crafty, 197.parser, 252.eon,
253.perlbmk, 254.gap, 255.vortex, 256.bzip2, 300.twolf
Compiling Binaries
  Building 164.gzip ref base vc7.x87.exe default
  Building 175.vpr ref base vc7.x87.exe default
  Building 176.gcc ref base vc7.x87.exe default
  Building 181.mcf ref base vc7.x87.exe default
  Building 186.crafty ref base vc7.x87.exe default
  Building 197.parser ref base vc7.x87.exe default
  Building 252.eon ref base vc7.x87.exe default
  Building 253.perlbmk ref base vc7.x87.exe default
Error with make 'specmake build > make.out 2> make.err': check file 'C:/spec/benchspec/cint2000/253.perlbmk/run/00000001/make.err'
Error with make!
*** Error building 253.perlbmk
  Building 254.gap ref base vc7.x87.exe default
  Building 255.vortex ref base vc7.x87.exe default
  Building 256.bzip2 ref base vc7.x87.exe default
  Building 300.twolf ref base vc7.x87.exe default
Setting Up Run Directories
  Setting up 164.gzip ref base vc7.x87.exe default: created

```

XIII. EJERCICIO INDIVIDUAL

Analiza la documentación relativa al conjunto de benchmarks SPEC CPU2000 que encontrarás en <http://www.spec.org/cpu2000/docs/>

Analiza las tablas de medidas correspondientes al benchmark SPEC CPU publicadas periódicamente por la organización SPEC. Comprueba qué tipo de CPU tienen los ordenadores del laboratorio y busca los benchmark SPEC para ese procesador. Si no existe ese tipo concreto de CPU busca el que te parezca más parecido y analiza los resultados que da SPEC.

Una vez encontrado y chequeadas las medidas de velocidad de base y de pico, visualiza el fichero de resultados correspondiente (p.e. en formato HTML) mediante el enlace habilitado para ello. Observar detalladamente los resultados de cada uno de los programas y analizar los diferentes factores que han

influido en los resultados obtenidos en dicha máquina, tales como frecuencia del procesador, memoria instalada, compilador utilizado, optimizaciones, etc.

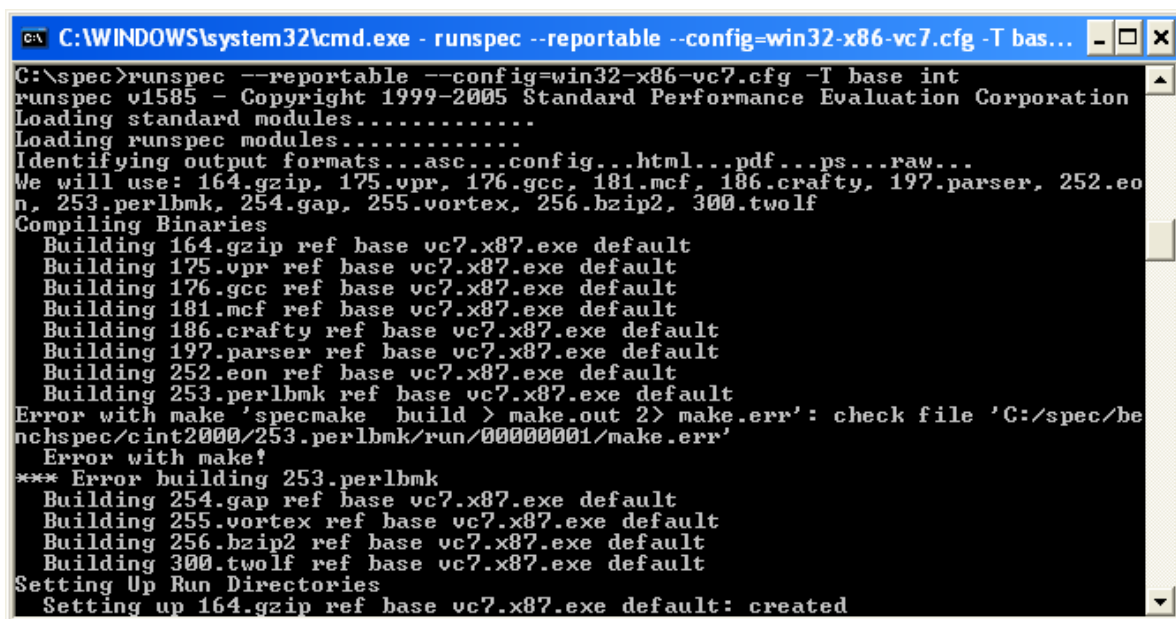
Ejecuta los benchmark de SPEC con el siguiente comando:

```
runspec --reportable --config=win32-x86-vc7.cfg -T base int
```

Reflexiona acerca de las siguientes preguntas:

¿Para qué sirve cada una de las opciones empleadas en el comando?

¿La ejecución de los benchmark ha dado los resultados que esperabas?, ¿por qué?



```
C:\WINDOWS\system32\cmd.exe - runspec --reportable --config=win32-x86-vc7.cfg -T bas...
C:\spec>runspec --reportable --config=win32-x86-vc7.cfg -T base int
runspec v1585 - Copyright 1999-2005 Standard Performance Evaluation Corporation
Loading standard modules.....
Loading runspec modules.....
Identifying output formats...asc...config...html...pdf...ps...raw...
We will use: 164.gzip, 175.vpr, 176.gcc, 181.mcf, 186.crafty, 197.parser, 252.eon, 253.perlbnk, 254.gap, 255.vortex, 256.bzip2, 300.twolf
Compiling Binaries
Building 164.gzip ref base vc7.x87.exe default
Building 175.vpr ref base vc7.x87.exe default
Building 176.gcc ref base vc7.x87.exe default
Building 181.mcf ref base vc7.x87.exe default
Building 186.crafty ref base vc7.x87.exe default
Building 197.parser ref base vc7.x87.exe default
Building 252.eon ref base vc7.x87.exe default
Building 253.perlbnk ref base vc7.x87.exe default
Error with make 'specmake build > make.out 2> make.err': check file 'C:/spec/benchspec/cint2000/253.perlbnk/run/00000001/make.err'
Error with make!
*** Error building 253.perlbnk
Building 254.gap ref base vc7.x87.exe default
Building 255.vortex ref base vc7.x87.exe default
Building 256.bzip2 ref base vc7.x87.exe default
Building 300.twolf ref base vc7.x87.exe default
Setting Up Run Directories
Setting up 164.gzip ref base vc7.x87.exe default: created
```

Utilizando los datos proporcionados en la página de SPEC y las métricas de velocidad y productividad, considera modelos similares del mismo fabricante, frecuencias de procesador similares y con 2, 4, 8 y 16 núcleos respectivamente. Compáralos y reflexiona acerca de cómo varían las métricas con el número de núcleos y si varía el ranking al pasar de enteros a flotantes.