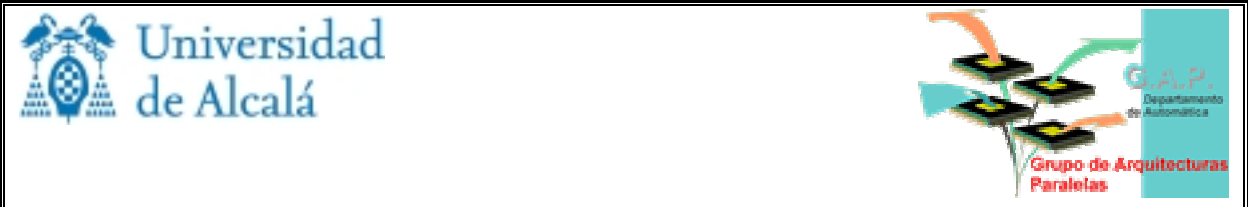


Propuesta de banco de pruebas para el repertorio de instrucciones x86 (subconjunto de 16 bits)

Informe técnico TR-UAH-AUT-GAP-2005-21-es

Rafael Rico

Departamento de Automática, Universidad de Alcalá, España

Noviembre 2005

English version:

Proposal of test-bench for the x86 instruction set (16 bits subset)

Technical Report TR-UAH-AUT-GAP-2005-21-en

Rafael Rico

Department of Computer Engineering, Universidad de Alcalá, Spain

---

Resumen:

Con el fin de evaluar el impacto de la arquitectura de repertorios de instrucciones sobre el procesamiento superescalar, aplicando un método matemático derivado de la teoría de grafos, se propone un conjunto de programas como banco de pruebas.

El ámbito de aplicación corresponde al procesamiento de números enteros. El repertorio de instrucciones seleccionado es el x86 debido a sus peculiares características respecto al paralelismo a nivel de instrucción.

Se presenta la metodología de obtención de trazas de ejecución y la carga de trabajo de cada uno de los programas seleccionados.

Finalmente, se realiza una caracterización de cada programa en base a su funcionalidad y a los recuentos de operaciones y operandos.

Palabras clave: Evaluación de arquitecturas de computadores, paralelismo a nivel de instrucción, arquitectura del repertorio de instrucciones.

Abstract:

With the purpose of evaluating the instruction set architecture impact on the superscalar processing, applying a mathematical method derived from the graph theory, a set of programs is proposed as test-bench.

The application corresponds to the integer processing. The instruction set selected is the x86 one due to its peculiar characteristics with respect to instruction level parallelism.

The methodology of obtaining of execution traces is presented and the work load of each one of the selected programs.

Finally, a characterization of each program is made on the basis of his functionality and to the counts of operations and operands.

Index words: Evaluation of computer architectures, instruction level parallelism, instruction set architecture.

1. Necesidad y objetivos

La arquitectura de los repertorios de instrucciones puede adecuarse a una variedad de especificaciones con el fin de optimizar su rendimiento respecto a algún aspecto. A lo largo de la historia de la computación se han impuesto diferentes criterios: minimizar el espacio de representación con el fin de generar ejecutables más pequeños, disminuir el desnivel semántico entre el ensamblador y los lenguajes de alto nivel para facilitar el trabajo de los compiladores, reducir el tiempo de compilación, alargar el tiempo de vida del repertorio, recortar el consumo energético, etc.

En la actualidad, cobra gran relevancia el estudio del comportamiento de la arquitectura de los repertorios de instrucciones en el entorno de procesamiento superescalar, universalmente adoptado por los procesadores de propósito general.

Por otra parte, la evaluación cuantitativa es un punto crucial en la investigación de arquitectura de computadores. La simulación se ha convertido en la primera herramienta de evaluación pero la construcción de buenos simuladores y la selección de cargas de trabajo adecuadas son tareas muy delicadas [4]. De manera alternativa, proponemos el análisis matemático de trazas de ejecución fundamentado en la adaptación de la teoría de grafos al caso del paralelismo a nivel de instrucción [5, 6].

Así pues, con el fin de abordar el estudio del impacto de la arquitectura de los repertorios de instrucciones sobre el procesamiento superescalar aplicando nuestro análisis matemático sobre trazas de ejecución se hace necesaria la selección de:

- un repertorio de instrucciones;
- un método de obtención de trazas; y de
- un banco de pruebas representativo.

2. El repertorio de instrucciones x86

En un entorno de procesamiento superescalar las pérdidas de rendimiento más notables son debidas a las dependencias de datos entre instrucciones. Estas dependencias de datos pueden ser inherentes al algoritmo encargado de procesar la información o a otras características tales como la limitación de recursos o a la propia arquitectura del repertorio de instrucciones.

Algunas de esas características del repertorio de instrucciones que provocan dependencias adicionales a las propias del cálculo son el uso dedicado de registros, los operandos implícitos (dependen de la operación y no son especificados por el programador), los registros utilizados para el cómputo de direcciones de memoria, los códigos de condición, etc.

La arquitectura del repertorio de instrucciones de la familia x86 es un ejemplo de todas estas características.

Con el fin de preservar la compatibilidad binaria con los procesadores previos, lo cual ha traido innegables beneficios, el repertorio de instrucciones x86 ha heredado características de diseño adecuadas a requerimientos del pasado pero claramente inapropiadas para el procesamiento superescalar. El diseño original del repertorio perseguía dos líneas maestras: minimizar el espacio de representación de las instrucciones y acortar la distancia entre los lenguajes de alto nivel y el lenguaje máquina para facilitar el proceso de compilación. Hoy en día estos requisitos no son importantes y, por otra parte, las limitaciones impuestas por esta arquitectura al ILP son comúnmente conocidas.

El repertorio x86 es el primer candidato para abordar este estudio ya que resulta tremendamente interesante analizar su comportamiento en el entorno superescalar.

A la vista de los resultados obtenidos por Huang y Peng, tanto para la plataforma DOS como para la Windows95 [3], y considerando la dificultad extra del entorno de 32 bits debido a la gran variedad de operandos, se ha tomado la decisión de restringir el análisis a aplicaciones de 16 bits operando en modo real DOS, y dejar para un futuro el estudio del repertorio completo de 32 bits.

3. Obtención de trazas de ejecución

La generación de trazas de ejecución se basa en el modo de ejecución paso a paso y en la modificación de la rutina de servicio a la interrupción número 1 con el fin de salvar el formato binario de la instrucción en curso. Para cada instrucción se salva al máximo número de bytes que puede ocupar, concretamente 6 bytes para el subconjunto de 16 bits del repertorio x86[1]. En muchos casos, salvar la máxima longitud posible del formato de la instrucción supone guardar mucha más información de la necesaria pero simplifica el procedimiento de trazado y puede ser utilizada posteriormente para analizar otros eventos, como por ejemplo, si los saltos son tomados o no.

No es necesario que los programas que se seleccionen para formar parte del banco de pruebas tengan disponible su código fuente ya que hemos construido las herramientas necesarias para poder trabajar a partir de ejecutables. Cuando no se dispone del código fuente, la imagen del ejecutable puede ser "inyectada" con un virus que provoca la entrada en ejecución paso a paso antes de comenzar el propio código del programa.

Queremos hacer constar que las trazas contienen la completa secuencia de ejecución para una carga de trabajo específica, es decir, las secuencias trazadas no son parciales sino que corresponden a la ejecución completa del programa. El objetivo es no trabajar con mezclas parciales de instrucciones que no representen el comportamiento real del programa. Sin embargo, las trazas no incluyen las instrucciones procesadas en las llamadas al sistema.

[1] Esto puede provocar algún error esporádico si se da el caso de que una instrucción de 6 bytes lleve prefijo. En ese caso, se necesitarían 7 bytes para salvar la instrucción completa.

4. Programas del banco de pruebas y carga de trabajo

Los programas del banco de pruebas se van a centrar en el procesamiento de números enteros. Se han buscado aplicaciones reales que ejecuten una variedad de funciones lo más representativa posible del procesamiento con enteros.

Concretamente, el banco de pruebas se compone de un total de 9 programas. Se han seleccionado varias utilidades del sistema operativo MS-DOS 5.0 (*comp*, *find* and *debug*), así como aplicaciones de uso común: un compresor-descompresor (*rar* versión 1.52) y un compilador de lenguaje C (*tcc* versión 1.0). También se ha utilizado uno de los programas de la plataforma SPEC95int95, por disponer del código fuente en lenguaje C, con la intención de poder realizar comparativas con imágenes compiladas bajo diferentes optimizaciones.

La carga de trabajo de estos programas ha sido seleccionada para obtener un recuento de instrucciones reducido con el fin de que los ficheros de traza sean manejables. A pesar de esto, las trazas representan casi 190 millones de instrucciones.

Algunos trabajos recientes advierten que la métrica y la carga de trabajo usadas para evaluar rendimientos pueden afectar a los resultados ya que son susceptibles de interactuar entre si [2]. En definitiva, la propia metodología puede incidir en la cuantificación.

En nuestro caso, el único requerimiento de las cargas de trabajo es que no generen procesamientos demasiado largos. El hecho de trazar la ejecución completa unido a que los programas seleccionados cubren un amplio abanico de tareas computacionales de enteros asegura disponer de un muestreo bastante real. En cuanto a la métrica, podemos estar seguros de su rigor ya que se trata de aplicar un método matemático sobre las dependencias de datos encontradas en las trazas. En ningún caso interactúa con las cargas de trabajo.

A continuación se describen los programas trazados así como las cargas de trabajo a las que han sido sometidos:

i. COMP. Esta utilidad del Sistema Operativo MS-DOS (versión 5.0) sirve para comparar los contenidos de dos ficheros en busca de diferencias. Si los ficheros tienen distinto tamaño se aborta la comparación puesto que evidentemente son distintos. Si tienen igual tamaño se compara carácter a carácter anotando las diferencias en *stdout*.

La carga de trabajo consiste en la comparación de dos ficheros de 35Kbytes entre los que se han preparado 10 diferencias:

```
C:\>COMP fichero1.txt fichero2.txt /A
```

ii. DEBUG. Es un programa depurador de ficheros ejecutables que permite ensamblar y desensamblar código, volcar el contenido de la memoria, ver y dar valor a los registros del procesador, buscar cadenas de caracteres, ejecutar código en modo paso a paso, etc.

La carga de trabajo que se ha ensayado para obtener la traza consiste en pasarle a la aplicación un fichero ejecutable y pedirle, posteriormente en línea de comandos, que desensamble 32Kbytes de código:

```
C:\>debug fichero.exe
-u cs:0 l7fff
0D45:0000 BA9718        MOV     DX, 1897
0D45:0003 8EC2          MOV     ES, DX
0D45:0005 FA            CLI
0D45:0006 8ED2          MOV     SS, DX
0D45:0008 BC109E        MOV     SP, 9E10
0D45:000B FB            STI
0D45:000C B87822        MOV     AX, 2278
...
```

iii. FIND. Es una utilidad del sistema operativo MS-DOS que busca una cadena de texto en un archivo y presenta en stdout las líneas en las que se ha encontrado dicha cadena. Representa una tarea de comparación entre enteros de tamaño byte.

La traza corresponde a la ejecución de la utilidad haciéndole buscar una cadena corta (sólo 4 caracteres) sobre un fichero de texto de 108.725 bytes:

```
C:\>FIND /N “data” fichero.txt
```

iv. GO T. Es el programa GO del conjunto de programas de prueba SPECint95. Ejecuta el juego de 'go' contra si mismo. El procesamiento tiene una gran parte de búsqueda de patrones así como de lógica de anticipación. Como suele suceder en este tipo de programas, hasta un tercio del tiempo de ejecución se consume en rutinas de manejo de datos, como se puede comprobar en el trabajo de A. Fernández [7].

Puesto que en este caso está disponible el código fuente, se han realizado dos compilaciones con optimizaciones totalmente opuestas con el fin de evaluar el posible impacto del proceso de compilación: una optimización en tamaño y otra en velocidad.

Se ha utilizado un compilador no específicamente diseñado para generar código superescalar: *Borland C++* versión 4.0. Se genera una imagen para la plataforma MS-DOS con las optimizaciones oportunas y sin incluir información de depuración en la imagen del ejecutable. Las operaciones en coma flotante se realizan mediante emulación, es decir, todo el código ejecuta operaciones enteras.

La presente es una optimización en tamaño (flags de compilación –O1 –Os –G).

Se ha procurado que la traza no fuera excesivamente grande ya que, para este programa, una traza más grande no aporta una mezcla de instrucciones distinta sino una mayor diferencia de tiempos de ejecución. Las trazas grandes hacen que el análisis sea excesivamente pesado y los ficheros que las contienen difíciles de manejar. Por ello se han buscado unos argumentos que generen pocos pasos en el juego. Concretamente los argumentos “30 4” dan lugar a 11 pasos (movimientos) antes de abandonar el programa.

```
C:\>go 30 4
  1 B B4
  2 W D3
  3 B A2
  4 W C2
  5 B B3
  6 W C1
  7 B D2
  8 W C3
  9 B C4
 10 W pass
 11 B pass
Game over
```

v. GO V. Es la traza correspondiente al mismo programa fuente anterior y con la misma carga de trabajo pero cuya compilación ha sido optimizada en velocidad (flags –O2 –Ot –Ox –G).

vi. RAR C. Es la versión 1.52 de agosto de 1994 de un compresor/descompresor que puede trabajar tanto en línea de comandos como bajo una *shell* diseñada como ventana completa en modo texto.

La carga de trabajo con la que se ha generado la traza ha sido funcionando como compresor.

```
C:\>rar a –m5 –std compri.rar @listado
```

Se comprimen los ficheros que se indican en el archivo 'listado' enviando los mensajes a *stdout* (*switch –std*) y con el máximo nivel de compresión (*switch –m5*). El listado contiene 17 ficheros con un total de 543.437 bytes.

vii. RAR D. Corresponde al mismo programa que en el caso anterior pero trabajando como descompresor.

La traza se ha generado a partir del fichero que se comprimió previamente (*compri.rar*) y que ocupa 147.489 bytes. La orden en línea de comando es:

```
C:\>rar e –std compri.rar
```

viii. SORT. Es una utilidad del Sistema Operativo MS-DOS que ordena la información de entrada y la manda a *stdout*, a un archivo o a un dispositivo. La información de entrada puede ser un fichero o la salida de un comando del propio Sistema Operativo. Representa una tarea de comparación entre enteros de tamaño byte dentro de un algoritmo de ordenación.

En nuestro caso la entrada es un fichero de texto con una relación de nombres y apellidos organizados en columnas separadas por tabuladores. La salida se realiza sobre un fichero de texto organizado de la misma forma pero ordenado de manera ascendente:

```
C:\>SORT <Relación.txt >Ordenado.txt
```

ix. TCC. Este programa es el compilador en línea de comandos de TURBO C++ 1.0. de 1990.

Se ha compilado un fichero fuente sencillo con las opciones por defecto:

```
#include <stdio.h>

int main(void)
{
   char ch;

   printf("Input a character:");
   ch = getc(stdin);
   printf("The character input was: '%c'\n", ch);
   return 0;
}
```

```
C:\>Tcc prueba.c
```

5. Caracterización de los programas de prueba

La caracterización de los programas trazados se fundamenta en el recuento de operaciones, operandos, modos de direccionamiento, etc. Concretamente se han hecho los siguientes recuentos:

- distribución de operaciones;
- distribución de modos de direccionamiento;
- uso de registros;
- accesos a memoria;
- uso de operandos implícitos;
- llamadas a procedimientos;
- distribución de saltos; y
- transferencias con la pila

En base a estos recuentos, se han determinado las siguientes medidas:

- tiempo de ejecución secuencial;
- CPI secuencial;
- tamaño medio del bloque básico, y
- número medio de instrucciones por bloque básico que modifican el registro de estado.

La información obtenida es muy abundante. La tabla 1 nos puede dar idea de su volumen habida cuenta la cantidad de instrucciones trazadas.

Tabla 1. Programas del banco de pruebas y sus trazas.

programa	fichero de traza	tamaño del fichero de traza	nº de instrucciones procesadas
COMP	comp.des	4.193.220	689.866
DEBUG	debug.des	53.123.640	8.071.335
FIND	find.des	39.226.350	6.119.641
GO T	got.des	207.013.518	30.636.605
GO V	gov.des	204.972.426	30.290.351
RAR C	rar_c.des	735.383.556	98.244.064
RAR D	rar_d.des	110.220.936	14.782.924
SORT	sort.des	2.568.774	271.989
TCC	tcc.des	6.828.900	1.010.078
TOTAL instrucciones procesadas			**190.116.853**

Para presentar con claridad nuestro análisis se van a mostrar primero los resultados promedio del uso de operaciones de todo el banco de pruebas. Seguidamente, vamos a describir individualmente cada una de las aplicaciones trazadas en función de sus propias distribuciones de operaciones y en contraste con los valores promedio. En esta parte, se hará una breve referencia al uso de operandos.

Es conveniente hacer una pequeña aclaración. Puesto que trabajamos con trazas, todo salto condicional está resuelto en la secuencia anotada en el fichero de traza. Es decir, el manejo de trazas asume una predicción perfecta de saltos permitiendo construir secuencias de código tan largas como se desee.

a. Resultados promedio

Con objeto de presentar los datos de la manera más ordenada posible, vamos a clasificar las instrucciones. El juego de instrucciones x86 puede dividirse en las siguientes categorías:

- operaciones de transferencia: relativas al movimiento de datos
- operaciones de proceso: aritméticas y lógicas
- bifurcaciones: instrucciones de alteración de la secuencia de ejecución
- operaciones con cadenas: relacionadas con el manejo de tiras de caracteres
- operaciones de control: resto de instrucciones (manejo de banderas de estado y de control, parada, etc.)

Presentamos seguidamente la gráfica que ilustra cómo es la distribución promedio por categorías para el conjunto de programas de prueba.

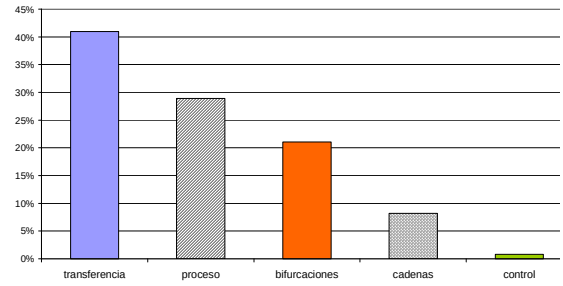


Fig. 1. Distribución media de operaciones por categorías.

La distribución es coherente con los resultados expuestos en el trabajo de Adams y Zimmerman acerca del uso de instrucciones en el 8086 [1].

La categoría más usada es la de las operaciones de transferencia con algo más del 40%. La instrucción predominante en este grupo es el MOV con más de un 30% de ocurrencias en valor medio.

El segundo grupo por volumen de uso es el de las operaciones de proceso que incluye tanto las aritméticas como las lógicas. En él destaca la instrucción CMP (8,31%) seguida de ADD (cerca del 5%) e INC (4,21%). El conjunto de las tres da lugar a más de la mitad de las ocurrencias del grupo. El uso de operaciones lógicas es menor. Entre ellas destacan SHL/SAL, OR y XOR.

Las bifurcaciones se llevan un 20% del total de operaciones siendo las más usadas las condiciones buscando igualdad (o cero) y su contraria (desigualdad o no cero).

Hasta aquí los datos son prácticamente iguales a los del trabajo citado anteriormente. Se aprecian algunas diferencias en la columna representando las instrucciones de manejo de cadenas. Su altura es mayor que la presentada en dicho artículo debido a dos aspectos. En primer lugar nosotros hemos incluido en el banco de pruebas un programa que hace un uso intensivo de este tipo de instrucciones (SORT) y en segundo lugar la métrica utilizada con las operaciones de cadena ha sido diferente. Las instrucciones de cadenas pueden ser modificadas por un prefijo de repetición que provoca su ejecución continúa hasta el final de la cadena al modo como lo haría si estuviera dentro de un bucle. Pues bien, Adams y Zimmerman las contabilizan una sóla vez y miden la longitud de la cadena procesada mientras que nosotros contabilizamos todas y cada una de las ejecuciones de la instrucción, tal y como lo haríamos si pertenecieran a un bucle, sin determinar el tamaño de la tira de caracteres.

Finalmente, las instrucciones de control tienen un peso muy pequeño en el conjunto de la distribución.

Para conocer un poco mejor cada uno de los programas del banco de pruebas vamos a presentar la distribución de operaciones por categorías para cada uno de ellos.

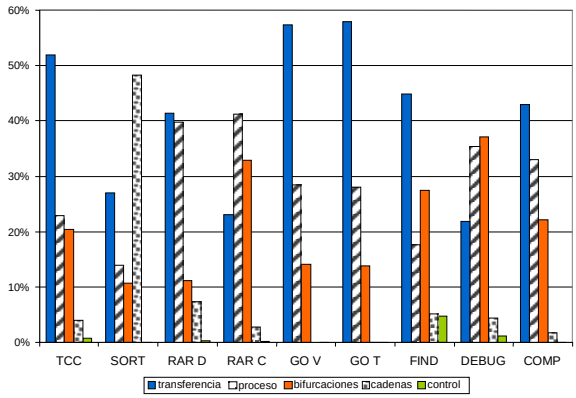


Fig. 2. Distribución de operaciones por categorías para cada programa.

A simple vista se hace patente que el conjunto de programas de prueba es muy heterogéneo, no encontrándose un patrón común al que se ajusten las distribuciones. Se alejan ostensiblemente de la distribución media los siguientes programas: SORT, por su excesivo uso de operaciones sobre cadenas; RAR C, debido al predominio de las instrucciones de proceso; FIND, con un significativo porcentaje de saltos; y DEBUG con un número muy pequeño de transferencias.

6. Caracterización individual

Seguidamente pasamos a realizar una caracterización de cada uno de los programas usados como banco de pruebas a la luz de su evaluación dinámica.

Concretamente, se presentan los recuentos de instrucciones ordenados por categorías y las 25 más usadas tabuladas en orden descendente de frecuencia porcentual. Se indica el parcial acumulado y se han sombreado las instrucciones responsables del 90% de las ejecuciones. Se ofrecen las instrucciones más usadas de cada categoría de una manera gráfica.

Como consecuencia del recuento se presenta el número total de instrucciones ejecutado, el número de subrutinas, las llamadas al sistema, los saltos condicionales y se ha calculado el número de bloques básicos y su tamaño medio.

Respecto a los datos, se muestra cierta información introductoria que será ampliada y

discutida en secciones posteriores: los accesos a registros y los accesos a memoria así como sus porcentajes sobre el total de instrucciones procesadas. Los accesos a registros se entienden como direccionamiento directos explícitos, es decir, no se tienen en cuenta los accesos ligados al código de operación y no indicados en el formato de la instrucción. Tampoco se incluyen las lecturas de registros provocadas por el cálculo de direcciones efectivas de memoria. En la contabilidad de los accesos a memoria, en esta parte, no se ha discriminado el modo de direccionamiento.

La ejecución simulada de las trazas permite obtener medidas de tiempos que también se dan en esta parte para el caso secuencial, es decir, el procesamiento cuando sólo se dispone de una unidad funcional. A partir de esta medida de tiempo y del número de instrucciones se calcula el CPI secuencial.

El esquema seguido para acometer la caracterización individual es el siguiente:

- describir qué hace el programa;
- explicar su perfil de operaciones a la luz de la tarea que realiza;
- determinar dónde se ubican los datos aunque sea de una manera gruesa (registros, memoria);
- contabilizar cuántas operaciones se realizan entre registros ya que normalmente son las más rápidas;
- observar la magnitud de las transferencias con la pila ya que se nos escapa de los recuentos de datos en memoria[2] e intentar decidir si corresponde a paso de parámetros o a escasez de registros;
- atender el uso de las llamadas a procedimientos ya sea a través de subrutinas[3] o llamadas al sistema[4];
- mirar el tamaño del bloque básico[5]; y finalmente,
- contrastar su rendimiento a través del CPI secuencial.

Respecto al trasiego con la pila hay que decir que tiene dos fuentes: el paso de parámetros a los procedimientos y la escasez de elementos de almacenamiento temporal (registros).

El modelo de programación del lenguaje C salva todos los registros del procesador en la pila en cada llamada a procedimiento. Es una manera de mantener la coherencia. Esto lo hace mediante instrucciones PUSH. Luego pasa los parámetros, realiza el procedimiento, devuelve el valor de retorno y, finalmente, reintegra los registros salvados anteriormente mediante instrucciones POP.

El retorno de procedimientos RET *n* ajusta el puntero de pila al lugar que se indique (*n*) de modo que no se usa POP para soslayar parámetros o variables locales.

PUSH y POP también tienen un uso combinado en la secuencia del código debido a la falta de espacio en el banco de registros y a su uso dedicado.

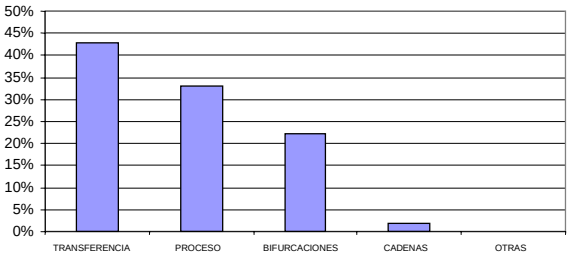
En resumen, si la cantidad de PUSH y POP ejecutados es similar podemos concluir que no se debe a los parámetros de las subrutinas si no a la escasez de elementos de almacenamiento temporal.

a. Programa COMP

Evaluación dinámica	
Instrucciones	689.866
Bloques básicos	152.239
Subrutinas	565
Llamadas al sistema	39
Saltos condicionales	115.578
Accesos a registros	385.476
Accesos a memoria	506.830
Porcentaje de accesos a registros	55,88%
Porcentaje de accesos a memoria	73,47%
Instrucciones por bloque básico	4,53
Tiempo de ejecución secuencial* (seg.)	0,124536660
CPI secuencial	18,05

* 8086 a 100MHz

Distribución de instrucciones por categorías



categoría	**recuento**	**%**
transferencia	296.796	43,02%
proceso	227.723	33,01%
bifurcaciones	153.324	22,23%
		98,26%
cadenas	11.955	1,73%
otras	68	0,01%
		1,74%

	operación	%	acumulado
1	MOV	42,26%	42,26%
2	CMP	16,07%	58,33%
3	INC	15,84%	74,17%
4	JNB/JAE	10,24%	84,40%
5	JE/JZ	5,65%	90,05%
6	JMP	5,31%	95,36%
7	SCAS	0,66%	96,03%
8	STOS	0,65%	96,67%
9	JNE/JNZ	0,60%	97,27%
10	OR	0,35%	97,63%
11	PUSH	0,34%	97,97%
12	LODS	0,31%	98,28%
13	SUB	0,25%	98,53%
14	POP	0,21%	98,74%
15	DEC	0,21%	98,94%
16	LES	0,16%	99,11%
17	ADD	0,14%	99,25%
18	MOVS	0,11%	99,36%
19	LOOP	0,09%	99,45%
20	JS	0,09%	99,54%
21	CALL	0,08%	99,62%

[2] Las instrucciones PUSH y POP implican accesos a memoria de la misma manera que una instrucción que involucre un dato ubicado en memoria pero no se registran en esa contabilidad a través de los operandos.

[3] Recuento de las ocurrencias de la instrucción CALL.

[4] Recuento de las ocurrencias de la instrucción INT.

[5] Por definición, un bloque básico es el conjunto de instrucciones que se ejecutan en secuencia de manera que una vez ejecutada la primera, siempre se ejecutan todas las demás [7]. No es necesario que todos los bloques básicos tengan una instrucción de control (la última), sino que simplemente la primera instrucción sea el destino de algún salto. Aquí se ha igualado el número de bloques básicos con el de saltos condicionales (incluyendo bucles) más incondicionales ya que hacemos el recuento sobre trazas de ejecución.

22	RET	0,08%	99,69%
23	XCHG	0,04%	99,73%
24	CBW	0,03%	99,76%
25	SHL/SAL	0,02%	99,78%

↑ se han sombreado las instrucciones responsables del 90% de las ejecuciones

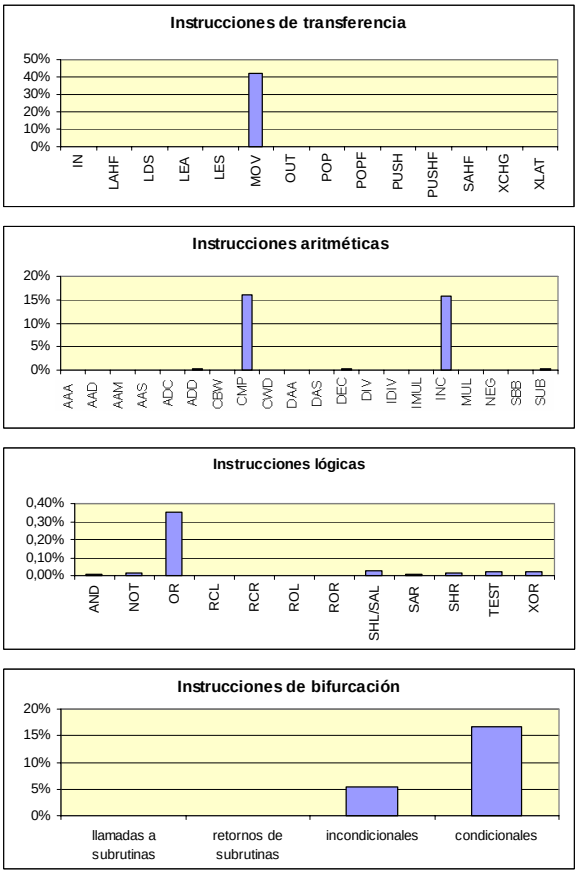


Fig. 3. Recuentos y estadísticas para COMP.

Este programa realiza una comparación carácter a carácter buscando desemparejamientos. Cuando los encuentra los muestra en pantalla. El núcleo del programa está compuesto por la comparación y la bifurcación condicional. La transferencia de datos resulta esencial dado que el juego del 8086 no permite realizar la comparación entre dos valores situados en posiciones de memoria. La actualización de los índices que recorren los listados de datos también va a tener un peso importante.

La distribución de instrucciones por categorías se ajusta a la media del banco de pruebas. El 90% de las instrucciones ejecutadas corresponde a tan sólo 5 operaciones: una de transferencia (MOV con el 43% aproximadamente); 2 de proceso (CMP e INC con algo más del 30%); y 2 saltos condicionales (con casi el 16%).

El tamaño del bloque básico es muy pequeño. Es lógico. En realidad el programa sólo realiza una comparación y salta a continuación en un bucle que se repite tantas veces como sea el tamaño de los ficheros a comparar. El resto de operaciones de la aplicación, sobre todo la lectura de los ficheros, se realizan mediante llamadas al sistema que no son trazadas.

Los accesos a datos en memoria tienen un porcentaje muy alto tanto respecto a la media del banco de pruebas como respecto al número total de accesos a datos en el programa. El porcentaje de accesos a registros de manera explícita está por debajo de la media y el total de operaciones entre registros es prácticamente despreciable. Las transferencias con memoria a través de la pila también son casi nulas lo que da idea de que el programa no ha apreciado escasez de registros.

Las llamadas a subrutinas y las llamadas al sistema son extremadamente escasas en relación con el número de instrucciones procesadas.

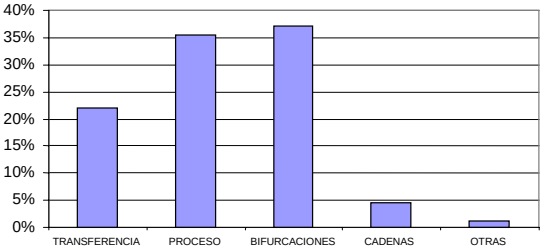
El CPI secuencial, es decir, el número de ciclos que se tarda en ejecutar una instrucción en promedio cuando sólo se cuenta con una unidad funcional, es el más alto de todos los programas del banco de pruebas y 6 ciclos por encima de la media que está en casi 12 ciclos. Este resultado parece lógico considerando el importante tráfico con memoria en acceso a datos. Por otro lado, el tiempo de vida medio de los datos con los que se opera es muy corto, ya que cambian los dos términos de la comparación en cada paso, haciendo difícil cualquier mejora de rendimiento.

b. Programa DEBUG

Evaluación dinámica	
Instrucciones	8.071.335
Bloques básicos	2.609.294
Subrutinas	195.159
Llamadas al sistema	13.148
Saltos condicionales	2.076.253
Accesos a registros	4.996.936
Accesos a memoria	1.221.895
Porcentaje de accesos a registros	61,91%
Porcentaje de accesos a memoria	15,14%
Instrucciones por bloque básico	3,09
Tiempo de ejecución secuencial* (seg.)	0,802615270
CPI secuencial	9,94

* 8086 a 100MHz

Distribución de instrucciones por categorías



categoría	**recuento**	**%**
transferencia	1.771.125	21,94%
proceso	2.852.731	35,34%
bifurcaciones	2.990.187	37,05%
		94,33%
cadenas	357.946	4,43%
otras	99.346	1,23%
		5,67%

	operación	%	Acumulado
1	JE/JZ	12,33%	12,33%
2	CMP	10,49%	22,82%

3	JNE/JNZ	10,38%	33,20%
4	MOV	9,45%	42,66%
5	JMP	6,60%	49,26%
6	PUSH	5,80%	55,06%
7	INC	5,72%	60,79%
8	POP	5,62%	66,41%
9	OR	5,28%	71,69%
10	DEC	3,91%	75,60%
11	TEST	3,49%	79,09%
12	ADD	3,06%	82,14%
13	STOS	2,50%	84,64%
14	CALL	2,42%	87,06%
15	RET	2,30%	89,36%
16	XOR	1,84%	91,20%
17	SCAS	1,46%	92,66%
18	JB/JNAE	1,35%	94,01%
19	JNB/JAE	0,92%	94,93%
20	XCHG	0,56%	95,48%
21	CLC	0,53%	96,02%
22	DIV	0,32%	96,34%
23	STC	0,30%	96,64%
24	SUB	0,29%	96,93%
25	JBE/JNA	0,28%	97,21%

† se han sombreado las instrucciones responsables del 90% de las ejecuciones

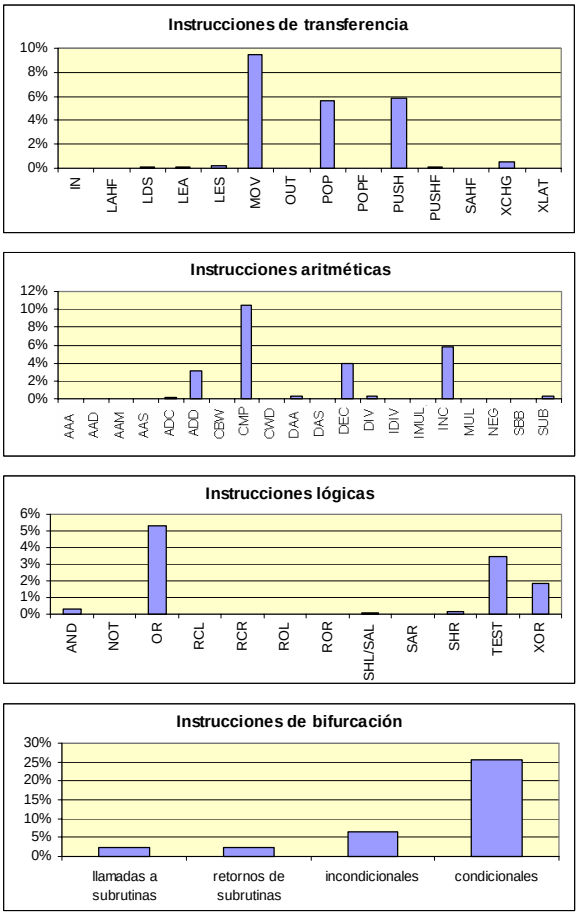


Fig. 4. Recuentos y estadísticas para DEBUG.

La carga de trabajo ha consistido en desensamblar un ejecutable. El proceso requiere la comparación de campos de bits con conjuntos de patrones siguiendo itinerarios diferentes en función de los resultados parciales.

La distribución de instrucciones no se ajusta a los promedios del banco de pruebas. Las más usadas son las bifurcaciones. Necesitamos 16 instrucciones para alcanzar un 90% del total procesado. Dentro de esta dispersión destaca el uso del condicional si cero y su contrario (si no cero) sumando más del 22%. Resulta importante el peso del salto incondicional (casi 7%). Esto corresponde a la compilación de estructuras del tipo *switch-case* utilizadas para implantar las búsquedas. Las bifurcaciones se completan con el uso de las llamadas a subrutinas y los retornos. El total de saltos a procedimientos es bastante grande en comparación con el resto de programas. Esto se justifica por que la decodificación de los campos del formato de una instrucción en lenguaje máquina es un proceso modular y repetitivo.

Las instrucciones de proceso ocupan el segundo puesto. Destaca, con un tercio del total, CMP usada para buscar los patrones de decodificación. Otras instrucciones de proceso muy empleadas son INC (casi un 6%), OR (algo más del 5%) y DEC (aproximadamente un 4%).

Finalmente, las transferencias están dominadas por MOV (9,45%), PUSH (5,8%) y POP (5,62%). El movimiento de datos con la pila es considerable y da idea de que el banco de registros se queda corto para las necesidades de almacenamiento temporal.

El porcentaje de datos que residen en memoria es muy pequeño (aproximadamente un 15% sobre el total de instrucciones). Asimismo, el porcentaje de datos que residen en registros y las operaciones que se realizan entre registros están ligeramente por debajo de la media.

Este es el programa con más llamadas al sistema superando al resto en varios órdenes de magnitud. Hay que considerar que la presentación de datos en pantalla es una parte fundamental del programa cosa que no sucede en el resto de aplicaciones trazadas. Aquí, por cada instrucción desensamblada se muestra en *stdout* su dirección de memoria en formato *base:desplazamiento*, el código máquina en hexadecimal y la cadena del lenguaje ensamblador ocupando más de 40 caracteres.

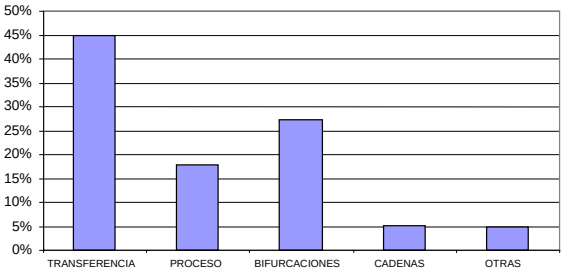
Hay que hacer notar que el tamaño del bloque básico es de los menores (3 instrucciones) y que el CPI secuencial es de los mejores de todo el banco de pruebas (9,94), debido al pequeño porcentaje de accesos a memoria.

c. Programa FIND

Evaluación dinámica	
Instrucciones	6.119.641
Bloques básicos	1.092.370
Subrutinas	291.984
Llamadas al sistema	151
Saltos condicionales	797.328
Accesos a registros	4.132.807
Accesos a memoria	979.543
Porcentaje de accesos a registros	67,53%
Porcentaje de accesos a memoria	16,01%
Instrucciones por bloque básico	5,60
Tiempo de ejecución secuencial* (seg.)	0,691962840
CPI secuencial	11,31

* 8086 a 100MHz

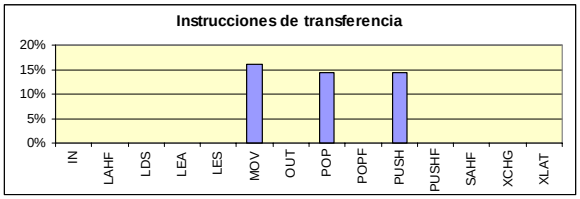
Distribución de instrucciones por categorías

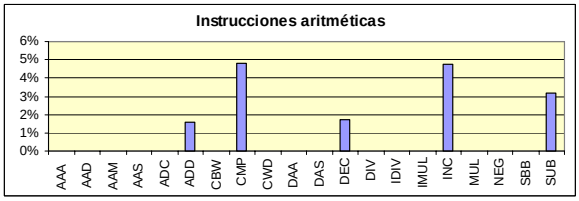


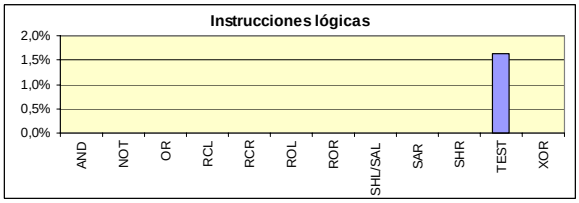
categoría	recuento	%
transferencia	2.749.843	44,93%
proceso	1.083.367	17,70%
bifurcaciones	1.676.339	27,39%
		90,03%
cadenas	318.271	5,20%
otras	291.821	4,77%
		9,97%

	operación	%	acumulado
1	MOV	16,16%	16,16%
2	PUSH	14,39%	30,55%
3	POP	14,39%	44,93%
4	CMP	4,84%	49,77%
5	JMP	4,82%	54,59%
6	CALL	4,77%	59,36%
7	RET	4,77%	64,14%
8	INC	4,76%	68,90%
9	CLC	4,73%	73,63%
10	SCAS	3,42%	77,05%
11	JNE/JNZ	3,33%	80,38%
12	JE/JZ	3,22%	83,60%
13	SUB	3,19%	86,79%
14	JNB/JAE	3,17%	89,95%
15	JCXZ	1,70%	91,66%
16	DEC	1,70%	93,36%
17	TEST	1,62%	94,98%
18	JB/JNAE	1,60%	96,59%
19	LODS	1,60%	98,19%
20	ADD	1,57%	99,76%
21	MOVS	0,10%	99,86%
22	STOS	0,08%	99,94%
23	STC	0,03%	99,98%
24	LOOP	0,01%	99,98%
25	XOR	0,00%	99,99%

↑ se han sombreado las instrucciones responsables del 90% de las ejecuciones







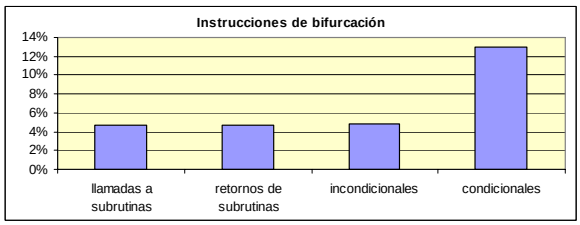


Fig. 5. Recuentos y estadísticas para FIND.

La traza estudiada de la utilidad FIND corresponde a la búsqueda de una cadena corta de 4 caracteres en un fichero de texto de 108.725 bytes. La comparación, en este caso, es con una cadena patrón. En cuanto se produce un fallo en la comparación se pasa a leer un nuevo conjunto de datos desde el fichero de entrada.

La distribución de instrucciones no se ajusta a los promedios obtenidos para el banco de pruebas debido al porcentaje de bifurcaciones. Dado el propósito de la utilidad no resulta extraño encontrar tantos saltos.

De entre las 25 instrucciones más usadas destacan las de transferencia en los primeros puestos. En primer lugar está MOV con un 16%, pero lo más interesante es ver a continuación el uso de PUSH y POP con un porcentaje idéntico en ambos casos de 14,39%. Sin duda se debe a la continua entrada y salida de datos del banco de registros por falta de espacio para almacenamiento temporal.

Las instrucciones de proceso dan lugar a casi el 18% de las operaciones ejecutadas con sólo tres instrucciones: CMP, lógicamente debido al carácter de la aplicación, con un 5%, INC y SUB con el 5% y el 3% respectivamente.

El resto de instrucciones responsables del 90% del trabajo del procesador corresponden a saltos. Hay un gran número de saltos incondicionales (casi un 5%) debido a que las comparaciones con la cadena patrón se suelen abortar, en la mayor parte de los casos, antes de alcanzar el final. Hay un gran número de llamadas/retornos a subrutinas (cerca del 5%): muy por encima del resto de programas del banco de pruebas. Finalmente, tres clases de saltos condicionales se llevan el 9% de ocurrencias.

Los datos residen básicamente en registros, siendo los emplazados en memoria los de menor porcentaje de todo el banco de pruebas junto con SORT. Sin embargo, hay que reseñar que el tráfico con memoria a través de la pila (15%) es incluso mayor que el derivado de los accesos explícitos a datos de memoria (13%).

Las llamadas al sistema son irrelevantes. El CPI secuencial y el tamaño medio del bloque básico están en el promedio del banco de pruebas.

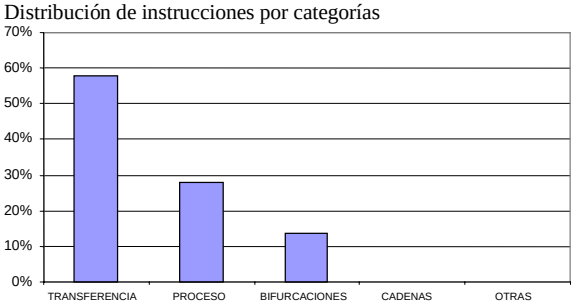
d. Programa GO optimizado en tamaño

Evaluación dinámica	
Instrucciones	30.636.605
Bloques básicos	3.829.543
Subrutinas	197.427
Llamadas al sistema	32

Saltos condicionales	2.972.717
Accesos a registros	32.214.654
Accesos a memoria	10.300.772
Porcentaje de accesos a registros	105,15%
Porcentaje de accesos a memoria	33,62%
Instrucciones por bloque básico	8,00
Tiempo de ejecución secuencial* (seg.)	3,782201650
CPI secuencial	12,35

* 8086 a 100MHz

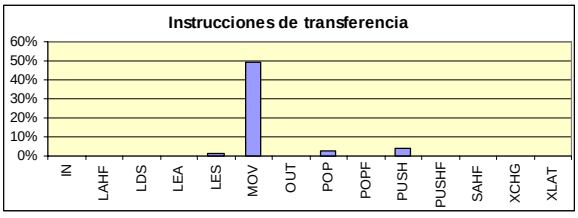
Distribución de instrucciones por categorías

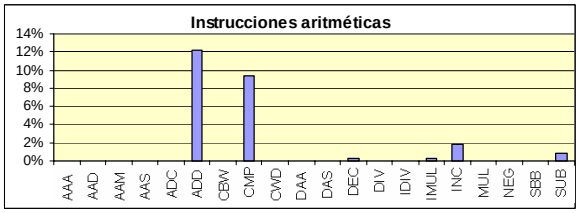


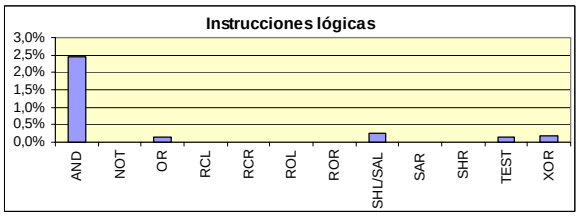
categoría	recuento	%
transferencia	17.772.338	58,01%
proceso	8.610.893	28,11%
bifurcaciones	4.224.388	13,79%
		99,91%
cadenas	9.117	0,03%
otras	19.869	0,06%
		0,09%

	operación	%	acumulado
1	MOV	49,66%	49,66%
2	ADD	12,20%	61,86%
3	CMP	9,34%	71,20%
4	PUSH	4,14%	75,34%
5	JNE/JNZ	3,87%	79,22%
6	JE/JZ	2,88%	82,10%
7	JMP	2,80%	84,89%
8	POP	2,50%	87,39%
9	AND	2,46%	89,86%
10	INC	1,88%	91,74%
11	LES	1,46%	93,20%
12	JNLE/JG	0,91%	94,11%
13	SUB	0,87%	94,98%
14	JL/JNGE	0,86%	95,84%
15	CALL	0,64%	96,48%
16	RETF	0,64%	97,13%
17	JNL/JGE	0,58%	97,71%
18	JLE/JNG	0,56%	98,27%
19	DEC	0,35%	98,61%
20	LEA	0,25%	98,86%
21	SHL/SAL	0,24%	99,09%
22	IMUL	0,22%	99,32%
23	XOR	0,17%	99,49%
24	TEST	0,16%	99,65%
25	OR	0,16%	99,82%

↑ se han sombreado las instrucciones responsables del 90% de las ejecuciones







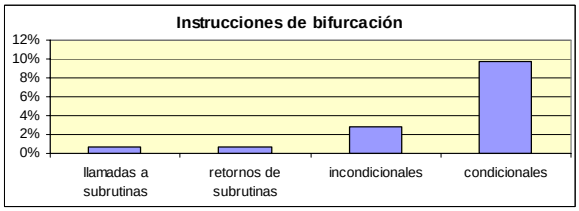


Fig. 6. Recuentos y estadísticas para GO (optimizado en tamaño).

La cuarta traza corresponde al programa GO de la *suite* SPECint95 compilado con la opción de optimizar en tamaño (se refiere a la imagen estática del ejecutable). Se comprueba que ahorra espacio: 596.602 bytes frente a los 599.018 bytes del mismo código compilado con la optimización en velocidad, aunque la diferencia es ridícula (menos de 3.000 bytes).

La distribución de la frecuencia de uso por categorías se ajusta a la media del banco de pruebas siendo despreciable la cantidad de instrucciones de manejo de cadenas y las de la clase otras. Es el programa que más instrucciones de transferencia utiliza alcanzando casi el 60%. Sólo 10 instrucciones se llevan el 90% de las ejecutadas y de ellas destaca el uso de MOV con un 50%. Los accesos a la pila con PUSH y POP suben un 7% más la frecuencia de las transferencias.

Las instrucciones de proceso suman algo más del 28% gracias a la ejecución de ADD (12,20%), CMP (9,34%), AND (2,46%) e INC (casi 2%).

Las instrucciones de salto están representadas por los condicionales si cero y si no cero y por el salto incondicional.

Los accesos a datos en memoria están en la media aunque hay que destacar el trasiego con la pila debido a PUSH y POP. El uso de operandos en registros está muy por encima de la media siendo importante la cantidad de instrucciones que operan entre registros. Todo indica que una arquitectura con más registros de propósito general, capaces de salvar datos temporales sin necesidad de acudir a memoria, optimizaría el rendimiento del programa.

El número de llamadas a subrutinas no es grande y las llamadas al sistema son despreciables frente a las instrucciones ejecutadas. El CPI secuencial está en la media y el tamaño del bloque básico por encima.

Los resultados obtenidos son consistentes con los presentados en el trabajo de Agustín Fernández [7] en el que hace una revisión de los programas del SPEC95 ejecutados sobre una arquitectura Alpha:

Tabla 2. Datos comparativos para GO sobre dos arquitecturas.

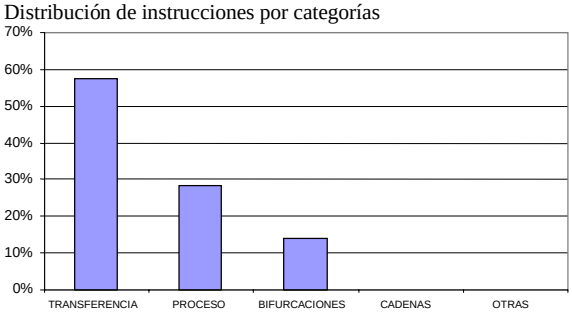
8086	concepto	Alpha [7]
58,01%	instrucciones de transferencia	50,42%
28,11%	operaciones de proceso	37,58%
13,79%	operaciones da salto	12,00%
12%	bloques básicos (sobre operaciones)	14%
0,64%	subrutinas (sobre operaciones)	0,86%
8	operaciones por bloque básico	6,9
MOV		LDx/STx
ADD	códigos de operación más usados	ADDx
CMP		CMP
JNE		BNE

e. Programa GO optimizado en velocidad

Evaluación dinámica	
Instrucciones	30.290.351
Bloques básicos	3.881.186
Subrutinas	197.032
Llamadas al sistema	21
Saltos condicionales	2.982.710
Accesos a registros	33.017.158
Accesos a memoria	8.995.946
Porcentaje de accesos a registros	109,00%
Porcentaje de accesos a memoria	29,70%
Instrucciones por bloque básico	7,80
Tiempo de ejecución secuencial* (seg.)	3,478782710
CPI secuencial	11,48

* 8086 a 100MHz

Distribución de instrucciones por categorías



categoría	recuento	%
transferencia	17.390.698	57,41%
proceso	8.604.322	28,41%
bifurcaciones	4.275.243	14,11%
		99,93%
cadenas	280	0,00%
otras	19.808	0,07%
		0,07%

	operación	%	acumulado
1	MOV	49,38%	49,38%
2	ADD	12,56%	61,94%
3	CMP	8,80%	70,74%
4	JNE/JNZ	4,77%	75,51%
5	PUSH	3,98%	79,49%
6	JMP	2,97%	82,46%
7	AND	2,49%	84,95%
8	JE/JZ	2,07%	87,01%
9	POP	2,05%	89,06%
10	INC	1,94%	91,00%
11	LES	1,77%	92,78%
12	JNLE/JG	0,90%	93,68%
13	OR	0,85%	94,53%
14	JL/JNGE	0,77%	95,30%
15	JNL/JGE	0,72%	96,02%
16	CALL	0,65%	96,67%
17	RETF	0,65%	97,32%
18	JLE/JNG	0,58%	97,90%
19	SUB	0,54%	98,44%
20	DEC	0,35%	98,79%
21	SHL/SAL	0,24%	99,03%
22	IMUL	0,23%	99,25%
23	LEA	0,22%	99,48%
24	XOR	0,21%	99,68%
25	TEST	0,17%	99,85%

† se han sombreado las instrucciones responsables del 90% de las ejecuciones

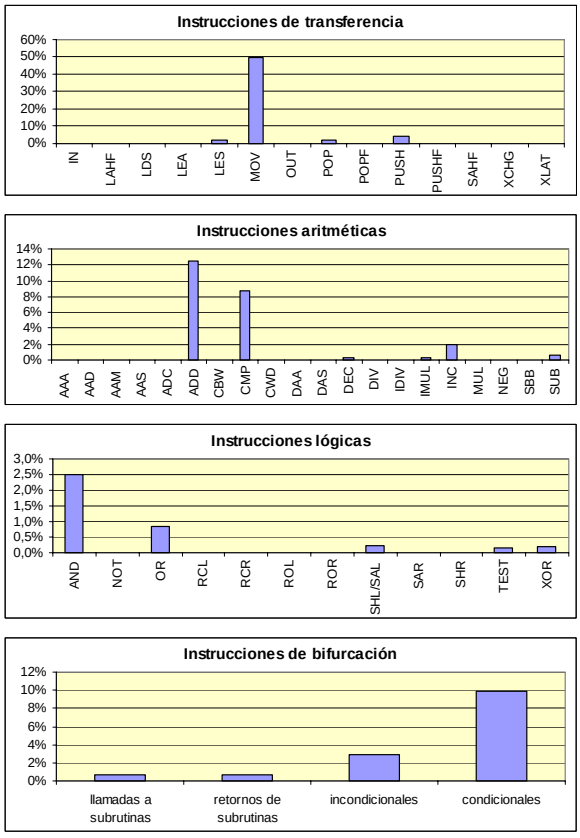


Fig. 7. Recuentos y estadísticas para GO (optimizado en velocidad).

Esta traza corresponde a la ejecución del mismo código fuente que en el caso anterior pero compilado con la opción de optimizar en velocidad. Se observa que la optimización ha sido doblemente efectiva puesto que se han ejecutado menos instrucciones (alrededor de un 1% menos) y además el CPI secuencial se ha reducido en casi un ciclo respecto a la versión anterior (11,48 frente al 12,35). Todo esto con una imagen del fichero ejecutable algo más grande. Esto demuestra que un ahorro estático de memoria no trae ninguna ventaja en tiempo de ejecución. La opción de optimizar en tamaño es algo heredado del pasado, cuando la memoria era un bien escaso dentro de los bloques de un computador, que ahora no implica una mejora en el rendimiento.

Los comentarios que se pueden hacer son muy similares a los de la traza previa: la distribución de frecuencia de uso es prácticamente igual a la del programa gemelo, las instrucciones responsables del 90% de las ejecuciones son las mismas, etc. Quizá podemos hablar de diferencias sutiles.

Se observa que el uso de la instrucción CMP ha descendido ahora en medio punto porcentual, es decir, se hacen menos comparaciones. También se hace un uso ligeramente menor de los intercambios con la pila. Crece el uso de datos en registros y de operaciones entre registros y disminuyen los accesos a datos de memoria. En definitiva, se ha mejorado el rendimiento minimizando las comparaciones y las transferencias con memoria.

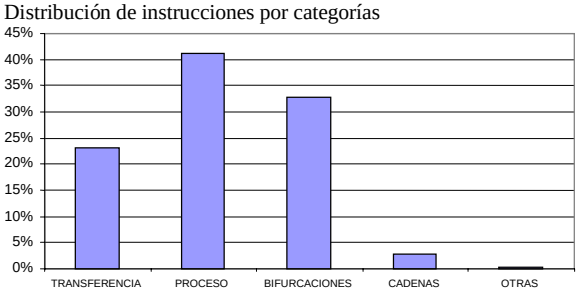
El tamaño del bloque básico es algo menor que en la traza gemela ya que el total de saltos es similar pero el número de instrucciones procesado es más pequeño.

f. Programa RAR comprimiendo

Evaluación dinámica	
Instrucciones	98.244.064
Bloques básicos	32.234.587
Subrutinas	18.127
Llamadas al sistema	751
Saltos condicionales	31.391.554
Accesos a registros	71.641.112
Accesos a memoria	30.680.803
Porcentaje de accesos a registros	72,92%
Porcentaje de accesos a memoria	31,23%
Instrucciones por bloque básico	3,05
Tiempo de ejecución secuencial* (seg.)	9,347562283
CPI secuencial	9,51

* 8086 a 100MHz

Distribución de instrucciones por categorías



categoría	**recuento**	**%**
transferencia	22.649.984	23,05%
proceso	40.437.561	41,16%
bifurcaciones	32.270.837	32,85%
		97,06%
cadenas	2.730.412	2,78%
otras	155.270	0,16%
		2,94%

	operación	%	acumulado
1	MOV	21,71%	21,71%
2	JE/JZ	19,90%	41,61%
3	SHL/SAL	12,72%	54,33%
4	CMP	10,23%	64,56%
5	JB/JNAE	5,77%	70,33%
6	XOR	3,73%	74,06%
7	DEC	3,49%	77,55%
8	TEST	2,45%	80,01%
9	JNB/JAE	2,33%	82,33%
10	ADD	2,23%	84,57%
11	JNE/JNZ	2,17%	86,74%
12	CMPS	2,00%	88,73%
13	SUB	1,75%	90,48%
14	AND	1,56%	92,04%
15	SHR	1,21%	93,25%
16	JMP	0,86%	94,11%
17	INC	0,80%	94,91%
18	XCHG	0,67%	95,58%
19	LOOP	0,62%	96,19%
20	STOS	0,60%	96,79%
21	JLE/JNG	0,33%	97,12%
22	OR	0,32%	97,44%
23	NOT	0,32%	97,77%
24	LDS	0,32%	98,09%
25	JNBE/JA	0,31%	98,40%

† se han sombreado las instrucciones responsables del 90% de las ejecuciones

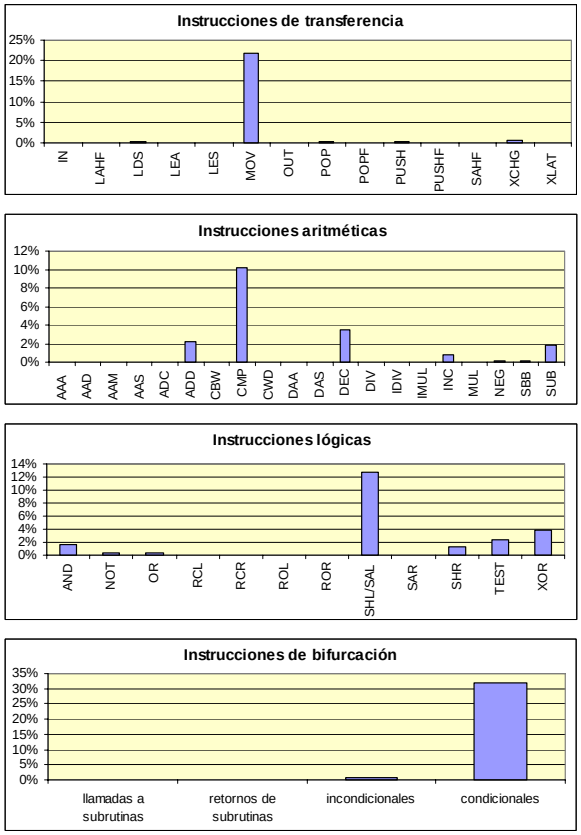


Fig. 8. Recuentos y estadísticas para RAR comprimiendo.

Esta traza corresponde a la ejecución de RAR con una carga de trabajo como compresor.

La distribución de instrucciones no se ajusta al promedio ya que prevalecen las de proceso y las bifurcaciones sobre las transferencias. El 90% de las operaciones procesadas lo soportan 13 instrucciones y de ellas la primera es el MOV que recopila prácticamente todas las transferencias (cerca del 22%). No se hace uso de la pila como almacenamiento temporal ya que PUSH y POP tienen un porcentaje de uso despreciable.

Las operaciones de proceso están representadas sobre todo por el desplazamiento a izquierda (SAL) con casi un 13% y las comparaciones con más de un 10%. Otras instrucciones aritméticas o lógicas son: XOR (3,73%), DEC (3,49%), TEST (2,45%), ADD (2,23%) y SUB (1,75%).

Las bifurcaciones se concretan en 4 tipos distintos de saltos condicionales ocupando el saltar si cero el segundo lugar entre las instrucciones más usadas con casi un 20%.

El uso de operandos en memoria está en la media mientras que el uso de operandos en registros está por

debajo de la misma. Sin embargo, destaca que el porcentaje de operaciones entre registros es muy elevado.

Las llamadas al sistema y a subrutinas son prácticamente nulas.

El CPI secuencial es el mejor de todo el banco de pruebas seleccionado en esta investigación con un 9,51. Seguramente esto se debe a la gran cantidad de operaciones entre registros que son las que menos ciclos consumen en esta arquitectura y al reducido trasiego con la pila por falta de registros.

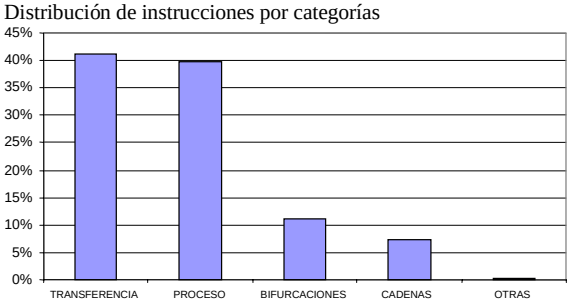
El tamaño del bloque básico también es el menor de entre todas las trazas del banco de pruebas.

g. Programa RAR descomprimiendo

Evaluación dinámica	
Instrucciones	14.782.924
Bloques básicos	1.629.633
Subrutinas	14.735
Llamadas al sistema	254
Saltos condicionales	1.398.728
Accesos a registros	15.633.667
Accesos a memoria	5.006.326
Porcentaje de accesos a registros	105,75%
Porcentaje de accesos a memoria	33,87%
Instrucciones por bloque básico	9,07
Tiempo de ejecución secuencial* (seg.)	1,552791730
CPI secuencial	10,50

* 8086 a 100MHz

Distribución de instrucciones por categorías



categoría	**recuento**	**%**
transferencia	6.106.813	41,31%
proceso	5.875.812	39,75%
bifurcaciones	1.659.101	11,22%
		92,28%
cadenas	1.093.980	7,40%
otras	47.218	0,32%
		7,72%

	operación	%	acumulado
1	MOV	39,01%	39,01%
2	XOR	8,95%	47,96%
3	SUB	6,23%	54,19%
4	CMP	4,56%	58,75%
5	ADD	4,47%	63,22%
6	SHR	4,38%	67,60%
7	STOS	3,72%	71,32%
8	SHL/SAL	3,56%	74,88%
9	MOVS	3,43%	78,31%
10	JNB/JAE	1,94%	80,25%
11	AND	1,83%	82,08%
12	INC	1,82%	83,89%
13	DEC	1,59%	85,48%
14	JMP	1,56%	87,04%
15	LOOP	1,50%	88,54%
16	XCHG	1,42%	89,96%
17	JNE/JNZ	1,33%	91,29%
18	JNBE/JA	1,25%	92,54%
19	JS	1,15%	93,69%
20	JE/JZ	1,10%	94,79%
21	SBB	1,04%	95,84%
22	JB/JNAE	0,72%	96,56%
23	ADC	0,60%	97,16%
24	JBE/JNA	0,46%	97,62%
25	PUSH	0,40%	98,02%

↑ se han sombreado las instrucciones responsables del 90% de las ejecuciones

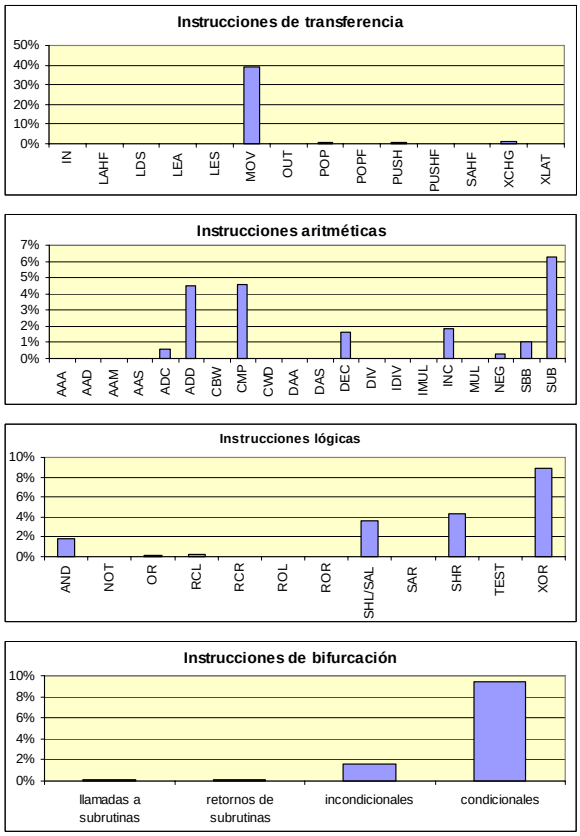


Fig. 9. Recuentos y estadísticas para RAR descomprimiendo.

La séptima traza se ha construido haciendo trabajar a RAR en descompresión. Se ha tomado como entrada el fichero comprimido anteriormente y se han restaurado los ficheros contenidos. Resulta llamativo como la tarea de comprimir ejecuta mucha más instrucciones que la de descomprimir (casi 100 millones frente a unos 15). La justificación está en que, mientras que la descompresión tiene una salida determinista, única, la compresión es una tarea heurística, basada en algoritmos de prueba.

El perfil de uso es totalmente distinto al anterior. Se ajusta a la media con un porcentaje de operaciones de proceso algo mayor de lo normal.

Entre las 25 instrucciones más usadas hay hasta 6 saltos condicionales distintos pero sin un peso apreciable. La cantidad de instrucciones responsables del 90% del procesamiento también es considerable. Todo ello da idea de que la distribución de instrucciones está muy dispersa.

La primera instrucción es el MOV con un 39% del total y se lleva el conjunto del peso de las transferencias.

Las operaciones aritmético/lógicas comprenden hasta 9 instrucciones con más del 35% del peso total.

Las bifurcaciones están representadas por una variedad de instrucciones entre las que cabría mencionar el uso de LOOP por su rareza en el conjunto del banco de pruebas.

Hay que destacar el 7% de instrucciones de manejo de cadenas.

Este programa es el que más operaciones realiza entre registros y uno de los que utiliza más datos residentes en registros. El uso de operandos de memoria está en la media. El tráfico con la pila es muy pequeño. Deducimos que el número de registros resulta suficiente.

Tiene un buen CPI secuencial. Contribuye a ello el gran número de operaciones rápidas (las que se hacen entre registros) aunque se ve atemperado por un uso de operandos en memoria ligeramente superior al del mismo programa en su uso como compresor.

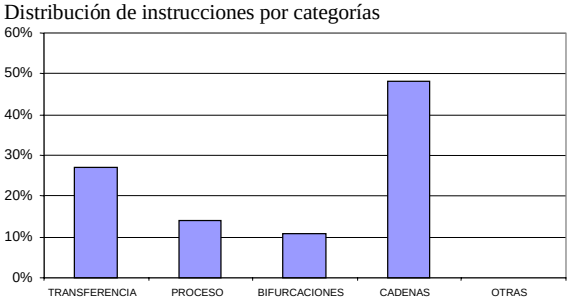
El tamaño del bloque básico es uno de los mayores de los programas de prueba con 9 instrucciones. Tanto las llamadas al sistema como a procedimientos son despreciables.

h. Programa SORT

Evaluación dinámica	
Instrucciones	271.989
Bloques básicos	29.144
Subrutinas	9
Llamadas al sistema	15
Saltos condicionales	28.802
Accesos a registros	130.653
Accesos a memoria	32.792
Porcentaje de accesos a registros	48,04%
Porcentaje de accesos a memoria	12,06%
Instrucciones por bloque básico	9,33
Tiempo de ejecución secuencial* (seg.)	0,028361530
CPI secuencial	10,43

* 8086 a 100MHz

Distribución de instrucciones por categorías



categoría	**recuento**	**%**
transferencia	73.362	26,97%
proceso	38.008	13,97%
bifurcaciones	29.162	10,72%
		51,67%
cadenas	131.352	48,29%
otras	105	0,04%
		48,33%

	operación	%	acumulado
1	MOVS	44,75%	44,75%
2	MOV	11,76%	56,50%
3	PUSH	6,00%	62,50%
4	POP	6,00%	68,50%
5	ADD	4,50%	73,00%
6	XLAT	3,22%	76,22%
7	CMP	3,15%	79,37%
8	SUB	3,05%	82,42%
9	JNBE/JA	2,93%	85,36%
10	SCAS	1,89%	87,25%
11	INC	1,71%	88,96%
12	LODS	1,61%	90,56%
13	LOOPZ/LOOPE	1,61%	92,17%
14	JNE/JNZ	1,57%	93,74%
15	JE/JZ	1,50%	95,24%
16	OR	1,48%	96,73%
17	JB/JNAE	1,47%	98,20%
18	JNB/JAE	1,47%	99,66%
19	JMP	0,13%	99,79%
20	DEC	0,06%	99,85%
21	STOS	0,05%	99,89%
22	JCXZ	0,04%	99,93%
23	CLD	0,02%	99,95%
24	SHR	0,02%	99,97%
25	STD	0,02%	99,98%

↑ se han sombreado las instrucciones responsables del 90% de las ejecuciones

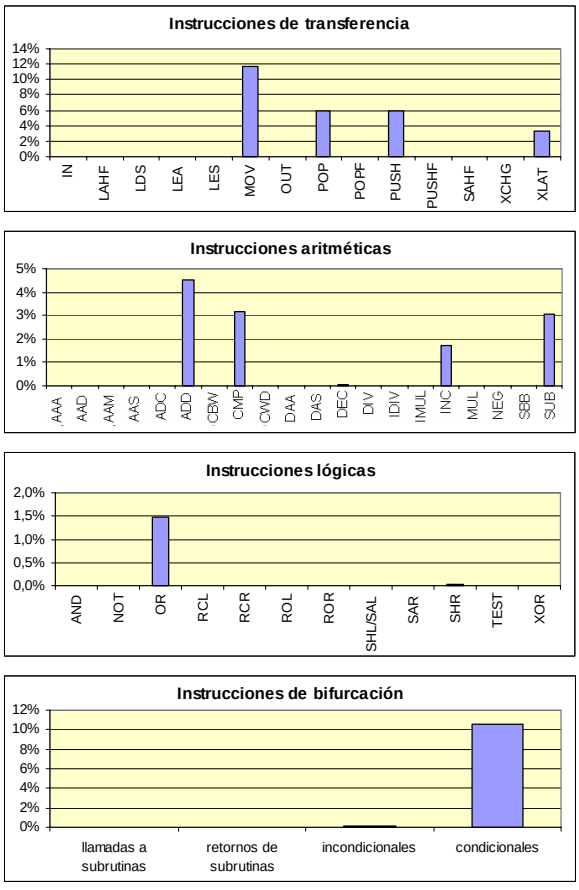


Fig. 10. Recuentos y estadísticas para SORT.

Esta traza pertenece a la ejecución de SORT sobre un fichero desordenado generando uno nuevo ordenado. Implica la aplicación de un algoritmo de ordenación sobre cadenas de caracteres y la copia ordenada de las mismas.

La distribución de instrucciones por categorías se sale del promedio en lo que a las operaciones de manejo de cadenas se refiere. De hecho, si el perfil promedio del banco de pruebas tiene una altura de algo más del 8% en esta clase, se debe a esta traza. Es bastante razonable pensar que si en lugar de

contabilizar cada repetición de una instrucción de cadena se hubiera contabilizado su aparición en la secuencia de código el porcentaje total se ajustaría mejor al resto de los programas.

Son 12 las instrucciones que acaparan el 90% de las operaciones ejecutadas. La primera es MOVS con un 45% aproximadamente. Otras instrucciones de cadena son SCAS y LODS con un 1,89% y un 1,61% respectivamente. Hay que pensar que una vez que las tiras de caracteres han sido ordenadas han de copiarse íntegramente en la salida. De ahí el porcentaje de ocurrencia de MOVS.

Las instrucciones de transferencia ocupan los 3 lugares más destacados después de MOVS. La segunda posición es para MOV con casi un 12% y le siguen PUSH y POP que se reparten a partes iguales otro 12%.

De entre las operaciones de proceso podemos destacar el uso de ADD (4,5%), CMP (3%), SUB (3%) e INC (casi 2%).

Los saltos se dan en un 10% del total de ocurrencias pero sólo aparece una instrucción de salto condicional (JNBE/JA con casi un 3%) entre las operaciones responsables del 90% de la carga de trabajo. En realidad las instrucciones de manejo de cadenas se repiten al modo como lo hacen las estructuras de control basadas en bucles. Por ello, el número de saltos condicionales implícitos es mayor.

La traza de SORT utiliza pocos datos, tanto en registros como en memoria, en comparación con los valores promedio del banco de pruebas. Sin embargo, éstos están implícitos en las instrucciones de manejo de cadenas. También se observa un tráfico de memoria considerable gracias a PUSH/POP sin tener un número significativo de llamadas a procedimientos. Todo el tráfico con la pila se debe, pues, a la escasez de almacenamiento. Las llamadas al sistema son inapreciables.

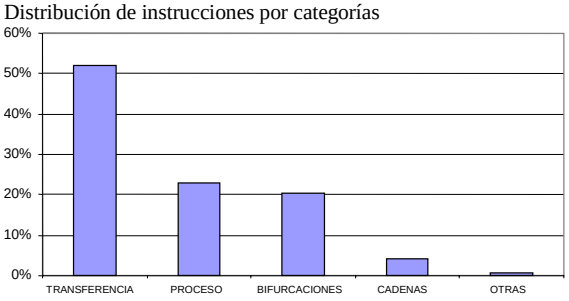
El CPI secuencial es bueno y el tamaño del bloque básico bastante grande si bien es cierto que si tomamos las repeticiones de las instrucciones de cadenas como saltos el bloque básico sería menor.

i. Programa TCC

Evaluación dinámica	
Instrucciones	1.010.078
Bloques básicos	159.463
Subrutinas	23.182
Llamadas al sistema	56
Saltos condicionales	124.397
Accesos a registros	770.882
Accesos a memoria	370.182
Porcentaje de accesos a registros	76,32%
Porcentaje de accesos a memoria	36,65%
Instrucciones por bloque básico	6,33
Tiempo de ejecución secuencial* (seg.)	0,142059500
CPI secuencial	14,06

* 8086 a 100MHz

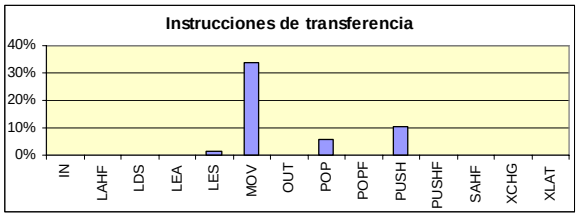
Distribución de instrucciones por categorías

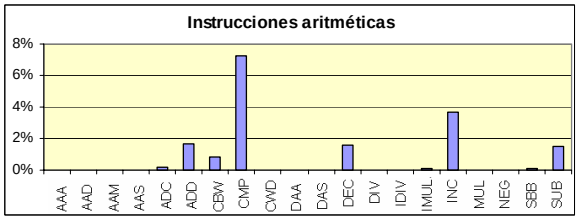


categoría	**recuento**	**%**
transferencia	524.947	51,97%
proceso	230.835	22,85%
bifurcaciones	205.812	20,38%
		95,20%
cadenas	40.892	4,05%
otras	7.592	0,75%
		4,80%

	operación	%	acumulado
1	MOV	33,87%	33,87%
2	PUSH	10,64%	44,51%
3	CMP	7,29%	51,80%
4	POP	5,78%	57,58%
5	JE/JZ	3,99%	61,58%
6	JNE/JNZ	3,79%	65,37%
7	INC	3,66%	69,02%
8	JMP	3,47%	72,49%
9	CALL	2,30%	74,79%
10	OR	2,10%	76,89%
11	RETF	1,74%	78,64%
12	ADD	1,63%	80,26%
13	DEC	1,59%	81,85%
14	SCAS	1,57%	83,42%
15	SUB	1,53%	84,95%
16	LES	1,41%	86,37%
17	STOS	1,37%	87,74%
18	JB/JNAE	1,31%	89,05%
19	LOOP	1,18%	90,24%
20	SHL/SAL	1,08%	91,32%
21	XOR	0,99%	92,31%
22	TEST	0,90%	93,21%
23	CBW	0,86%	94,07%
24	LODS	0,86%	94,94%
25	RET	0,55%	95,49%

† se han sombreado las instrucciones responsables del 90% de las ejecuciones





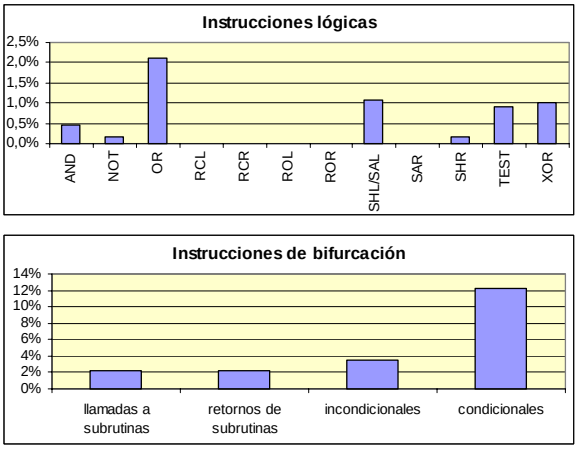


Fig. 11. Recuentos y estadísticas para TCC.

Es la traza proporcionada por la compilación mediante TCC de un programa fuente sencillo. El perfil de la distribución es similar al promedio con un peso de las bifurcaciones algo mayor tal y como debe ser en un compilador ya que la fase de análisis implica búsqueda de patrones y baterías de estructuras de control del tipo *switch-case*.

El conjunto de instrucciones responsable del 90% de la ejecución es grande lo que habla de una gran dispersión de operaciones. La primera instrucción es el MOV, con casi un 34% de frecuencia de uso, seguida de PUSH con un 10,64%. La instrucción POP se encuentra en cuarto lugar con cerca de un 6%.

Entre las instrucciones de proceso destaca CMP situada en tercer lugar con un 7,29% de frecuencia. Es lógico que esta operación se repita con frecuencia debido a las búsquedas de patrones. Otras operaciones de esta clase son INC, OR, ADD, DEC y SUB que suman alrededor del 10%.

Los saltos están representados principalmente por tres condicionales, por el salto incondicional y por un porcentaje alto de llamadas/retornos de subrutinas. Hay que destacar que LOOP se utiliza en más de un 1%.

Obsérvese como se utiliza la instrucción LES y el retorno de subrutina largo. El hecho se debe, sin duda, al tamaño del ejecutable (es el mayor después de GO) que implica el continuo cambio de segmento.

También se usan algunas instrucciones de manejo de cadenas.

Los operandos tanto en registro como en memoria se ajustan a los valores promedio del banco de pruebas. El número de operaciones entre registros, sin embargo, está por debajo de la media.

El tráfico con la pila es muy importante.

Las llamadas al sistema no son significativas.

El CPI secuencial no es muy bueno estando 2 ciclos por encima de la media. El tamaño del bloque básico está justo en el valor medio.

7. Referencias.

[1] T. L. Adams and R. E. Zimmerman, “An analysis of 8086 instruction set usage in MS DOS programs,” in *Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-III)*, April 1989, pp. 152 - 160.

[2] D. G. Feitelson. “Metric and Workload Effects on Computer Systems Evaluation,” *IEEE Computer*, vol. 36, 9, September, 2003.

[3] I. J. Huang and T. C. Peng, “Analysis of x86 Instruction Set Usage for DOS/Windows Applications and Its Implication on Superscalar Design,” *IEICE Transactions on Information and Systems*, Vol.E85-D, No. 6, pp. 929-939, June 2002. (SCI).

[4] K. Skadron, M. Martonosi, *D.* I. August, M. *D.* Hill, *D.* J. Hill and V. S. Pai. “Challenges in Computer Architecture Evaluation,” *IEEE Computer*, vol. 36, 8, August, 2003.

[5] R. Durán, R. Rico, “On Applying Graph Theory to ILP Analysis,” Technical Note TN-UAH-AUT-GAP-2005-01-en. Available at: http://atc2.aut.uah.es/~gap/

[6] R. Durán, R. Rico, “Quantification of ISA Impact on Superscalar Processing,” in *Proceedings of Eurocon2005*, November 2005.

[7] A. Fernández., “Un análisis cuantitativo del Spec95,” Informe técnico UPC-DAC-1999-12, Universidad Politécnica de Cataluña, Barcelona, 1999.