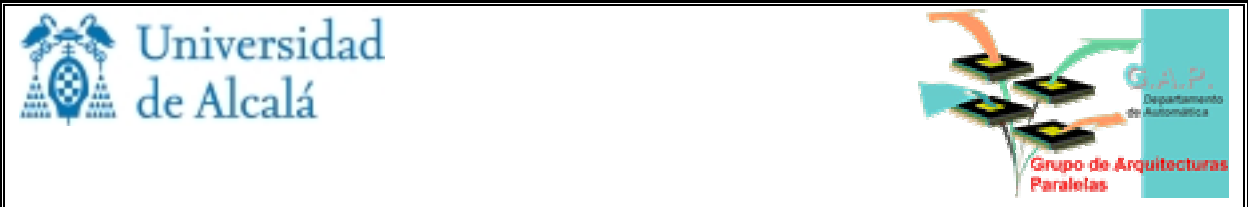

Análisis del uso de datos en el repertorio x86 (subconjunto de 16 bits)

Informe técnico TR-UAH-AUT-GAP-2005-22-es

Rafael Rico

Departamento de Automática, Universidad de Alcalá, España

Diciembre 2005

English version:

Analysis of x86 Data Usage (16 bits subset)

Technical Report TR-UAH-AUT-GAP-2005-22-en

Rafael Rico

Department of Computer Engineering, Universidad de Alcalá, Spain

---

Resumen:

Para estudiar el comportamiento de los repertorios de instrucciones en el entorno de procesamiento superescalar es necesario analizar el uso que se hace de los datos ya que el factor limitante más importante de la ejecución paralela son las dependencias de datos.

El presente informe técnico muestra la distribución del uso de los datos para el repertorio x86, subconjunto de 16 bits. El trabajo se ha realizado a partir de un banco de pruebas predefinido.

El estudio detallado del acceso a datos se ha organizado como sigue. Primero se analiza el uso explícito de registros, a continuación el uso implícito y finalmente se estudia el uso de las banderas de estado.

A partir de las distribuciones de uso de cada grupo se obtienen resultados cualitativos acerca de las fuentes más importantes de potenciales dependencias de datos.

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

Abstract:

To study the behavior of instruction sets in the superscalar setting to analyze the data usage is necessary because the main limiting factor to parallel execution is the data dependences among instructions.

This technical report shows the data usage distribution for x86 instruction set, 16 bits subset. The work has been done with a predefined test-bench.

The detailed study of data access has been organized as follows. First of all, the explicit register usage is analyzed, next the implicit usage and finally, the status flag usage is studied.

From usage distributions for each group, quantitative results about the most important sources of potential data dependences are obtained.

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

1. Introducción

Cuando se desea estudiar el comportamiento de los repertorios de instrucciones en el entorno de procesamiento superescalar es necesario analizar el uso que se hace de los datos.

El presente informe técnico muestra la distribución del uso de los datos para el repertorio x86, subconjunto de 16 bits. El trabajo se ha realizado a partir de un banco de pruebas cuya descripción en profundidad se puede conocer en el informe previo TR-UAH-AUT-GAP-21, titulado "*Propuesta de banco de pruebas para el repertorio de instrucciones x86 (subconjunto de 16 bits)*" [5].

2. Frecuencia de uso de instrucciones

Aunque no es el objetivo del informe, se ha realizado un estudio de la frecuencia de uso de las instrucciones agrupándolas por nemónico. Vamos a presentar nuestros resultados comparándolos con los obtenidos en 1989 por Adams y Zimmerman [1]. Como aseguran estos autores, aún hoy día, hay muy pocos estudios sobre trazas dinámicas del repertorio x86. En este informe se han trazado 190 millones de instrucciones frente a los 18 que trazaron ellos. La tabla 1 muestra las 25 instrucciones más usadas en promedio según ambos estudios.

A primera vista resulta claro que el conjunto de las instrucciones es similar en ambos casos. En los datos de 1989 se han resaltado sobre fondo gris las 6 instrucciones que han salido de la tabla de más usadas respecto a nuestros recuentos. La salida de LOOP era esperada ya que, como explica Randall Hyde, en la última versión de su libro sobre el lenguaje ensamblador [3], los compiladores han dejado de utilizar esta instrucción debido al uso dedicado que impone a través del contador CX: cuando se anidan varios bucles hay que hacer un continuo trasiego con la pila para salvar la coherencia de los índices. Sólo 3 aplicaciones del banco de pruebas usan LOOP y lo hacen con un peso próximo al 1,5%: RAR descomprimiendo, SORT y TCC (ver la tabla 2 con las 25 instrucciones más usadas para todas y cada una de las trazas).

También salen de la lista LES y LDS muy posiblemente debido a que nuestro banco de pruebas no cuenta con programas excesivamente grandes ni con áreas de datos muy voluminosas.

El resto de instrucciones salientes tiene más que ver con el perfil de los programas trazados que con criterios de compilación.

En nuestra tabla vemos que han entrado las instrucciones MOVS, JB/JNAE, AND, SCAS, STOS y CLC. De las tres instrucciones de manejo de cadenas incorporadas (MOVS, SCAS y STOS), la primera se debe a la traza de SORT mientras que las otras dos tienen un uso muy irregular en el resto de trazas. Respecto a esto también es conveniente hacer notar que nosotros hemos contabilizado todas las ocurrencias de las operaciones con cadenas mientras que los autores del trabajo previo sólo las contaban una vez (descartando el uso del prefijo) y median su longitud con objeto de calcular la longitud media tratada. Lógicamente nuestros recuentos son mucho mayores cuando las longitudes de las tiras de caracteres son grandes. Esta diversidad de contabilidades afecta algo a los porcentajes de uso de las operaciones.

En muchos casos los porcentajes de uso resultan similares (MOV, CMP, JNE/JNZ, SHL/SAL, SUB, XOR, DEC, OR) ya que corresponden a instrucciones básicas del repertorio. Por el contrario, resulta significativo que PUSH y POP han disminuido en porcentaje a la vez que han aproximado sus valores. Esto puede tener la explicación en una compilación optimizada para disminuir las transferencias con memoria a través de la pila y en un mejor aprovechamiento de los registros disponibles[1].

Hay que destacar como MOV es la operación más frecuentemente usada con gran diferencia. De todas las trazas solamente dos no la tienen en primera posición: DEBUG, en favor de los condicionales, y SORT, en favor de una transferencia orientada a cadenas.

La segunda operación es CMP, una operación aritmética (resta) que no escribe resultados. Este comportamiento tiene gran trascendencia ya que al no escribir en el operando destino no genera dependencias por datos explícitos[2], aunque si lo hace por implícitos[3]. Se usa para actualizar el registro de estado escribiendo en las banderas la información resultante de la comparación. Es ahí donde se establece la posible dependencia. Suele formar pareja con un salto condicional por lo que entre ella y la bifurcación no suele encontrarse ninguna otra instrucción que modifique el registro de estado.

Respecto a los saltos condicionales, vemos que tan sólo hay cuatro distintos que a su vez son complementarios dos a dos. En consecuencia, podemos decir que de los 16 códigos de operación correspondientes a condicionales en este repertorio, muchos de ellos podrían no estar. Algo similar ocurre con otras instrucciones: operaciones de corrección BCD, XCHG o el caso ya comentado de LOOP.

En la cola tenemos a SHR, que no sale de la lista gracias al uso que le da el programa RAR, y CLC, que debe su inclusión a FIND.

A continuación podemos ver una serie de gráficas comparativas de los siguientes aspectos a lo largo de todo el banco de pruebas: transferencias con la pila, llamadas a procedimientos, tiempos de ejecución secuencial y tamaños del bloque básico. En todas las gráficas se ha dibujado el valor promedio.

---

1 En muchas ocasiones una asignación óptima de registros produce el mismo resultado que contar con un número mayor.

2 Se considera dato explícito aquel que aparece en la instrucción.

3 Se considera dato implícito aquel que no aparece en el formato de la instrucción ya que está ligado al código de operación; no puede ser seleccionado por el programador.

Tabla 1. Listado de las 25 operaciones más frecuentemente usadas en promedio: a la izquierda según el artículo de Adams y Zimmerman [1] y a la derecha según los recuentos obtenidos en este informe.

	MEDIA (trabajo previo)		
	operación	**%**	**acumulado**
1	MOV	29,95	29,95
2	PUSH	9,25	39,2
3	CMP	7,94	47,14
4	POP	6,22	53,36
5	JNE/JNZ	4,52	57,88
6	JE/JZ	3,63	61,51
7	ADD	3,47	64,98
8	CALL	3,29	68,27
9	RET	3,17	71,44
10	JMP	2,3	73,74
11	LOOP	1,96	75,7
12	INC	1,95	77,65
13	OR	1,84	79,49
14	SUB	1,74	81,23
15	SHL/SAL	1,38	82,61
16	XOR	1,17	83,78
17	DEC	1,17	84,95
18	LES	1,13	86,08
19	TEST	1,04	87,12
20	JNB/JAE	0,84	87,96
21	LDS	0,74	88,7
22	SHR	0,75	89,45
23	RCR	0,72	90,17
24	RETF	0,69	90,86
25	JNBE	0,66	91,52

	MEDIA (resultados actuales)		
	operación	**%**	**acumulado**
1	MOV	30,36	30,36
2	CMP	8,31	38,67
3	JE/JZ	5,85	44,52
4	MOVS	5,45	49,97
5	PUSH	5,10	55,07
6	ADD	4,71	59,77
7	INC	4,24	64,01
8	POP	4,12	68,13
9	JNE/JNZ	3,53	71,66
10	JMP	3,17	74,83
11	JNB/JAE	2,28	77,11
12	SHL/SAL	1,99	79,10
13	SUB	1,97	81,07
14	XOR	1,77	82,83
15	DEC	1,47	84,30
16	JB/JNAE	1,36	85,67
17	CALL	1,22	86,89
18	OR	1,19	88,08
19	AND	1,01	89,09
20	SCAS	1,01	90,09
21	STOS	1,00	91,09
22	TEST	0,98	92,07
23	RET	0,87	92,94
24	SHR	0,66	93,60
25	CLC	0,59	94,19

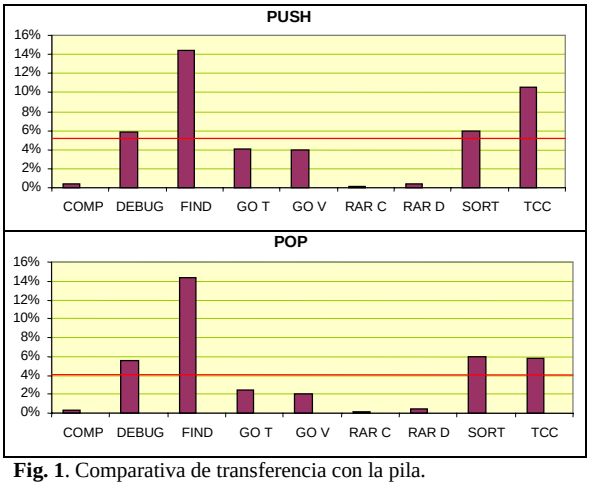

Fig. 1. Comparativa de transferencia con la pila.

Respecto a las transferencias con la pila, vemos como destaca FIND por la cantidad y TCC por la desigualdad entre las operaciones PUSH y POP. El caso de FIND denota escasez de registros lo que implica salvar en la pila datos temporales con objeto de cambiar el uso de los mismos. Esto concuerda con el uso dedicado de los registros que en secciones posteriores se describe. Respecto a TCC, la diferencia de porcentajes entre PUSH y POP se justifica en atención a dos empleos diferentes. El porcentaje de uso de POP y el equivalente de PUSH hacen referencia un banco de registros limitado. El exceso de uso de PUSH respecto a POP habla del paso de parámetros a subrutinas a través de la pila que no tiene su contraparte en retornos con POP.

En cuanto a las llamadas a procedimientos, observamos el comportamiento absolutamente heterogéneo en cuanto a subrutinas y la anomalía de DEBUG en las llamadas al sistema.

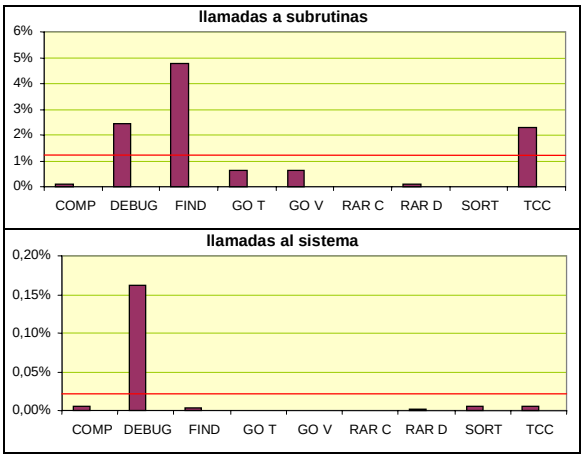

Fig. 2. Comparativa de llamadas a procedimientos.

El programa que más llamadas a subrutinas hace es FIND aunque eso no va acompañado de un de paso de parámetros importante, mientras que TCC sí pasa una cantidad importante de parámetros y COMP mucho mayor aún. Seguidamente, cuando hablemos del "uso explícito de registros", tendremos ocasión de describir con más detalle este aspecto.

La cantidad de llamadas al sistema que hace DEBUG se justifica por el hecho de que es el que tiene la mayor salida de datos por pantalla.

Tabla 2. Listado de las 25 operaciones más frecuentemente usadas por programa.

	COMP		DEBUG		FIND		GO T		GO V		RAR C		RAR D		SORT		TCC	
	operación	%	operación	%	operación	%	operación	%	operación	%	operación	%	operación	%	operación	%	operación	%
1	MOV	42,26	JE/JZ	12,33	MOV	16,16	MOV	49,66	MOV	49,38	MOV	21,71	MOV	39,01	MOVS	44,75	MOV	33,87
2	CMP	16,07	CMP	10,49	PUSH	14,39	ADD	12,20	ADD	12,56	JE/JZ	19,90	XOR	8,95	MOV	11,76	PUSH	10,64
3	INC	15,84	JNE/JNZ	10,38	POP	14,39	CMP	9,34	CMP	8,80	SHL/SAL	12,72	SUB	6,23	PUSH	6,00	CMP	7,29
4	JNB/JAE	10,24	MOV	9,45	CMP	4,84	PUSH	4,14	JNE/JNZ	4,77	CMP	10,23	CMP	4,56	POP	6,00	POP	5,78
5	JE/JZ	5,65	JMP	6,60	JMP	4,82	JNE/JNZ	3,87	PUSH	3,98	JB/JNAE	5,77	ADD	4,47	ADD	4,50	JE/JZ	3,99
6	JMP	5,31	PUSH	5,80	CALL	4,77	JE/JZ	2,88	JMP	2,97	XOR	3,73	SHR	4,38	XLAT	3,22	JNE/JNZ	3,79
7	SCAS	0,66	INC	5,72	RET	4,77	JMP	2,80	AND	2,49	DEC	3,49	STOS	3,72	CMP	3,15	INC	3,66
8	STOS	0,65	POP	5,62	INC	4,76	POP	2,50	JE/JZ	2,07	TEST	2,45	SHL/SAL	3,56	SUB	3,05	JMP	3,47
9	JNE/JNZ	0,60	OR	5,28	CLC	4,73	AND	2,46	POP	2,05	JNB/JAE	2,33	MOVS	3,43	JNBE/JA	2,93	CALL	2,30
10	OR	0,35	DEC	3,91	SCAS	3,42	INC	1,88	INC	1,94	ADD	2,23	JNB/JAE	1,94	SCAS	1,89	OR	2,10
11	PUSH	0,34	TEST	3,49	JNE/JNZ	3,33	LES	1,46	LES	1,77	JNE/JNZ	2,17	AND	1,83	INC	1,71	RETF	1,74
12	LODS	0,31	ADD	3,06	JE/JZ	3,22	JNLE/JG	0,91	JNLE/JG	0,90	CMPS	2,00	INC	1,82	LODS	1,61	ADD	1,63
13	SUB	0,25	STOS	2,50	SUB	3,19	SUB	0,87	OR	0,85	SUB	1,75	DEC	1,59	LOOPZ	1,61	DEC	1,59
14	POP	0,21	CALL	2,42	JNB/JAE	3,17	JL/JNGE	0,86	JL/JNGE	0,77	AND	1,56	JMP	1,56	JNE/JNZ	1,57	SCAS	1,57
15	DEC	0,21	RET	2,30	JCXZ	1,70	CALL	0,64	JNL/JGE	0,72	SHR	1,21	LOOP	1,50	JE/JZ	1,50	SUB	1,53
16	LES	0,16	XOR	1,84	DEC	1,70	RETF	0,64	CALL	0,65	JMP	0,86	XCHG	1,42	OR	1,48	LES	1,41
17	ADD	0,14	SCAS	1,46	TEST	1,62	JNL/JGE	0,58	RETF	0,65	INC	0,80	JNE/JNZ	1,33	JB/JNAE	1,47	STOS	1,37
18	MOVS	0,11	JB/JNAE	1,35	JB/JNAE	1,60	JLE/JNG	0,56	JLE/JNG	0,58	XCHG	0,67	JNBE/JA	1,25	JNB/JAE	1,47	JB/JNAE	1,31
19	LOOP	0,09	JNB/JAE	0,92	LODS	1,60	DEC	0,35	SUB	0,54	LOOP	0,62	JS	1,15	JMP	0,13	LOOP	1,18
20	JS	0,09	XCHG	0,56	ADD	1,57	LEA	0,25	DEC	0,35	STOS	0,60	JE/JZ	1,10	DEC	0,06	SHL/SAL	1,08
21	CALL	0,08	CLC	0,53	MOVS	0,10	SHL/SAL	0,24	SHL/SAL	0,24	JLE/JNG	0,33	SBB	1,04	STOS	0,05	XOR	0,99
22	RET	0,08	DIV	0,32	STOS	0,08	IMUL	0,22	IMUL	0,23	OR	0,32	JB/JNAE	0,72	JCXZ	0,04	TEST	0,90
23	XCHG	0,04	STC	0,30	STC	0,03	XOR	0,17	LEA	0,22	NOT	0,32	ADC	0,60	CLD	0,02	CBW	0,86
24	CBW	0,03	SUB	0,29	LOOP	0,01	TEST	0,16	XOR	0,21	LDS	0,32	JBE/JNA	0,46	SHR	0,02	LODS	0,86
25	SHL/SAL	0,02	JBE/JNA	0,28	XOR	0,00	OR	0,16	TEST	0,17	JNBE/JA	0,31	PUSH	0,40	STD	0,02	RET	0,55

Finalmente, en dos gráficas más podemos ver de un golpe los resultados calculados en rendimiento secuencial y en tamaño del bloque básico. Podemos adelantar una correlación entre CPI y accesos a memoria (representan un cuello de botella) de forma que el peor CPI, el de COMP, se corresponde con el mayor porcentaje de accesos a memoria, muy por encima de la media.

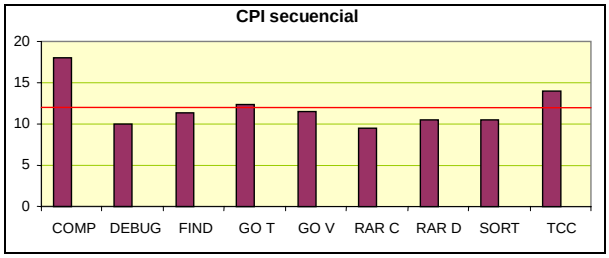

Fig. 3. Comparativa de rendimiento secuencial en CPI.

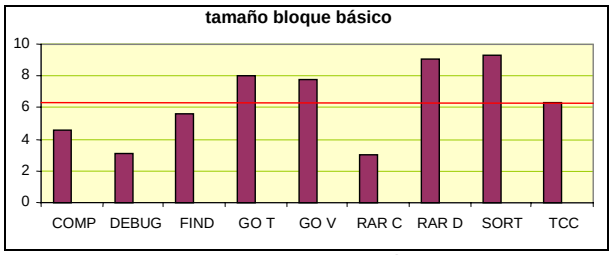

Fig. 4. Comparativa de tamaño del bloque básico.

a. Uso de instrucciones fuera del repertorio *Intel* 8086

Dado que los programas utilizados para generar las trazas no han sido compilados, salvo en el caso de GO, ya que no se disponía de los fuentes, se ha realizado un recuento de instrucciones no incluidas en el repertorio del *Intel* 8086 (el propio del procesador 8086, sin extensiones posteriores).

El resultado es que ninguno de los programas del banco de pruebas utiliza extensiones posteriores exceptuando el compilador TCC que lo hace en un 1% de ocasiones aproximadamente. Esto coincide con las mismas mediciones realizadas por Adams y Zimmerman [1] con la advertencia de que nuestros ejecutables son más modernos, pues corresponden a versiones posteriores, y por tanto más susceptibles de haberlas incorporado.

La conclusión es que las extensiones agregadas al juego del procesador 8086, especialmente en el 80386, sólo son aplicadas a la escritura de determinadas partes de los sistemas operativos (memoria virtual, protección de memoria, etc.) que no se encuentran en las aplicaciones.

b. Uso de prefijos

En otro orden de cosas, vamos a mencionar brevemente los recuentos en cuanto a prefijos. En el repertorio del 8086 tenemos prefijos de repetición de cadenas, prefijos de segmento y LOCK.

Los prefijos de repetición se usan para modificar las instrucciones de manejo de cadenas de manera que se repitan como si estuvieran encerradas en un bucle LOOP. Sería un bucle con una sóla instrucción, la de cadena[4]. Las trazas los han encontrado en una media del 2,62% salvo en el caso de SORT que hace un uso intensivo de estos prefijos alcanzando el 46,68%.

Los prefijos de segmento se usan para modificar la base por defecto empleada para calcular la dirección efectiva de memoria. Tienen que ver, por tanto, con los accesos a memoria. Hacemos notar que el registro base establece una dependencia entre instrucciones adicional, que el prefijo no hace más que convertir en explícita. Los prefijos de segmento se utilizan con una frecuencia promedio de algo más del 10% repartiéndose CS y ES la totalidad de las ocurrencias con un 4,25% y un 5,94% respectivamente.

Cada acceso a memoria implica el uso de un registro de segmento por defecto. La aparición de prefijos de segmento en el código no altera este hecho, tan sólo convierte en explícito lo que de manera natural está implícito.

El prefijo LOCK, cuya función es bloquear el acceso a recursos hardware compartidos por varios procesadores, no se ha encontrado en ninguna ocasión.

3. Estudio detallado del acceso a datos

En un entorno de procesamiento superescalar, el rendimiento final depende del algoritmo programado, de la impronta del compilador, de las limitaciones de la arquitectura y, en definitiva, de la mezcla de instrucciones con las que se ha creado la aplicación ya que las dependencias de datos que limitan la ejecución concurrente se manifiestan, en último término, entre los datos de las instrucciones. En este sentido, resulta interesante ahondar en cómo se accede a los datos ya que nos va a proporcionar información acerca de las limitaciones que impone la arquitectura tanto desde la realización física como desde el repertorio de instrucciones.

A continuación se ilustra gráficamente la distribución de los modos de direccionamiento en tres comparativas distintas: datos ubicados en registros explícitos, accesos a datos ubicados en memoria y operaciones efectuadas entre registros.

Vemos como la traza de COMP tiene el mayor porcentaje de accesos a memoria y el menor de operaciones entre registros. Evidentemente su CPI es el peor como consecuencia, en gran medida, de la latencia de los accesos a memoria. El mejor es el RAR comprimiendo, con un equilibrado uso de operaciones entre registros y accesos a memoria.

[4] Los prefijos de repetición encierran el mismo problema que el LOOP: hacen un uso dedicado del registro contador CX. Por su funcionalidad, no va a ser habitual que se aniden con otros bucles con lo que no hay que hacer transferencias con la pila para salvar el contador... Sin embargo, en muchas ocasiones sería más efectivo encerrar en un sólo bucle varias sentencias que, de esta manera, no es posible. El uso de estas operaciones con cadenas impone la distribución del proceso en varios bucles implícitos, cada uno de ellos con su prefijo de repetición. Hay que hacer notar además, que el prefijo de repetición de operaciones de manejo de cadenas da lugar a un bloque básico de tamaño unidad.

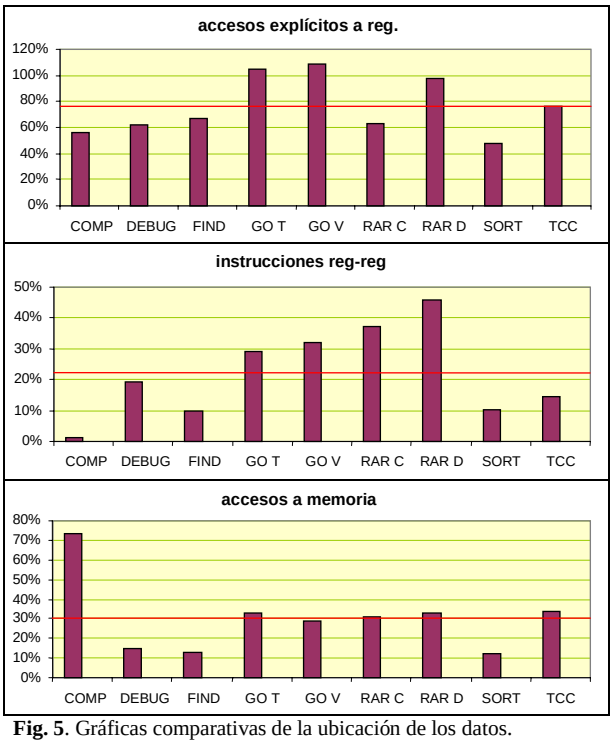

Fig. 5. Gráficas comparativas de la ubicación de los datos.

Queremos saber cómo se utilizan los registros bajo el modo de direccionamiento directo a registro, también queremos conocer cómo se accede a posiciones de memoria y de qué manera se calculan las direcciones efectivas y, finalmente, no podemos olvidar que hay un uso de datos implícito del que queremos evaluar el impacto sobre el rendimiento final de la máquina puesto que también es responsable de dependencias de datos. Vamos, pues, a enfocar el problema haciendo recuentos de uso de registros proporcionados de manera explícita en el formato de la instrucción tanto en acceso directo a registro como en acceso a memoria y luego pasaremos a estudiar los recuentos de uso implícito o ligado al código de operación.

Este estudio nos va a dibujar el panorama de las posibles dependencias de datos y, en consecuencia, la disposición que vamos a tener para aprovechar el procesamiento concurrente.

a. Uso explícito de registros

El uso de registros atiende a una doble funcionalidad: el procesamiento de datos y el cómputo de direcciones. El cómputo de direcciones, ya sean de memoria de datos o de acceso a pila, representa una carga computacional aunque no sea la estrictamente ligada al algoritmo programado.

En la figura 6 se dan las gráficas que ilustran el número de accesos a registros. En gris claro tenemos la cantidad de accesos en modo de direccionamiento directo a registro, es decir, aquellos utilizados para el procesamiento de datos. En gris oscuro se han representado acumulativamente los accesos a registros para calcular direcciones de memoria (modos de direccionamiento relativo a registro). Se han mostrado las lecturas. Ya que el repertorio de instrucciones es de dos direcciones, el operando destino (escrito) también se lee. Pretendemos describir el mapa de accesos más que si se usan en lectura o escritura.

Es llamativo observar como el uso de los registros se concentra en unos pocos en la mayor parte de los casos, especialmente en COMP, FIND y las dos versiones de GO. Esto es más evidente en el empleo de registros como punteros de memoria. Vemos que la variedad de registros usados para calcular direcciones es aún menor. Esta falta de versatilidad queda clara en la rejilla de codificación de modos de direccionamiento de memoria que presentamos a continuación:

Tabla 3. Calculo de la dirección efectiva de memoria en el formato de instrucción.

r/m	mod = 00	mod = 01	mod = 10
000	[BX]+[SI]	[BX]+[SI]+D8	[BX]+[SI]+D16
001	[BX]+[DI]	[BX]+[DI]+D8	[BX]+[DI]+D16
010	[BP]+[SI]	[BP]+[SI]+D8	[BP]+[SI]+D16
011	[BP]+[DI]	[BP]+[DI]+D8	[BP]+[DI]+D16
100	[SI]	[SI]+D8	[SI]+D16
101	[DI]	[DI]+D8	[DI]+D16
110	dirección directa	[BP]+D8	[BP]+D16
111	[BX]	[BX]+D8	[BX]+D16

El que la tabla anterior tenga tres columnas diferentes con desplazamiento 0, desplazamiento de 8 bits y desplazamiento de 16 bits es resultado del criterio de diseño del repertorio: ya que se pretende minimizar el espacio de representación y, por tanto, el tamaño de los formatos es variable en función de la cantidad de información requerida. En este caso, la codificación del formato debe indicar el número de bytes adicionales que se han de tomar del flujo de instrucciones para leer el desplazamiento. Así aseguramos que desplazamientos pequeños o nulos no ocupan memoria innecesariamente.

En definitiva, la tabla anterior se podría reducir a la Tabla 4 salvando el caso del direccionamiento directo a memoria. En ella se ha incluido el registro de segmento utilizado en cada caso como base.

Tabla 4. Registros implicados en el cálculo de direcciones. Se incluye el registro de segmento por defecto.

DSx16+[BX]+[SI]
DSx16+ [BX]+[DI]
SSx16+ [BP]+[SI]
SSx16+ [BP]+[DI]
DSx16+ [SI]
DSx16+ [DI]
SSx16+ [BP]
DSx16+ [BX]

Sólo cuatro registros se utilizan para calcular direcciones: BX y BP como base y SI, DI como índice, según la terminología del fabricante. A estos registros hay que añadir el uso implícito (salvo que se especifique un prefijo de manera explícita) de un registro de segmento por defecto: DS o SS dependiendo de la base y del índice.

El elevado número de registros involucrados en el cálculo de direcciones es una fuente potencial de dependencias entre datos.

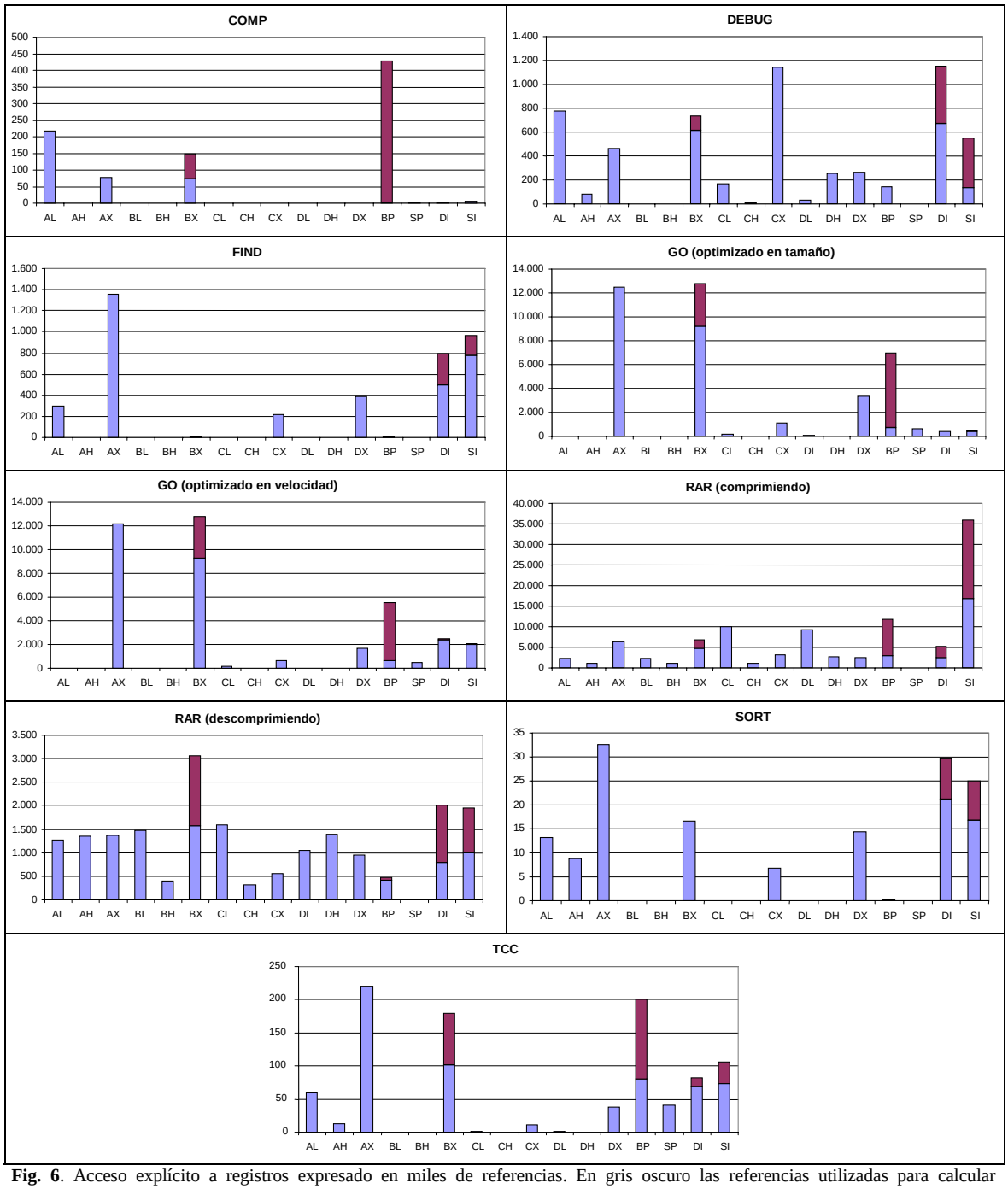

Fig. 6. Acceso explícito a registros expresado en miles de referencias. En gris oscuro las referencias utilizadas para calcular direcciones de memoria.

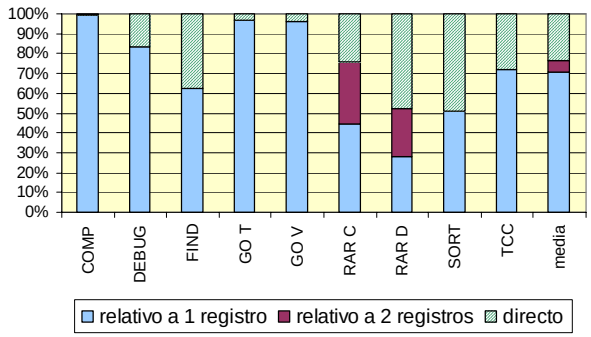

Fig. 7. Distribución del modo en el que se calcula la dirección efectiva en los accesos a memoria.

En la figura 7 se muestra la distribución de registros implicados en el cómputo de direcciones en los accesos a memoria sobre el total de accesos para cada programa del banco de pruebas y la media sobre todos ellos.

Se puede ver como lo más normal es utilizar un registro de los cuatro. Sin embargo, el programa RAR, tanto en compresión como en descompresión, utiliza dos registros (una de las bases y uno de los índices) en un importante porcentaje de los accesos. Esto implica aumentar las potenciales dependencias de datos que a la postre limitan el grado de paralelismo y por tanto el rendimiento. Obsérvese que la gráfica hace referencia a los registros utilizados como desplazamiento. Hay

que añadir, en cada caso, el registro de segmento que se haya usado como base. En definitiva, hay que contar con un registro más en cada caso.

Al mismo tiempo, debemos indicar que los accesos combinados a registros en tamaño palabra y en tamaño byte (AX, AL, AH, etc.) también son posibles fuentes de dependencias de datos ya que representan el mismo recurso físico.

La traza del programa COMP es la que menos registros utiliza: AX, BX y BP. El acumulador se utiliza en direccionamiento directo, el registro BP en direccionamientos a memoria (pila) y el BX se reparte para ambas funciones. Es fácil concluir que el grafo del código va a presentar muchas dependencias a través de estos tres registros limitando las oportunidades de ejecución concurrente.

El uso dedicado de registros y la elevada reutilización de los mismos implica un mayor grado de dependencias entre datos. En realidad dibuja un escenario de escasez de recursos físicos, aunque en términos absolutos podamos contar con una cantidad apreciable de elementos de almacenamiento temporal. Además, como señalan Adams y Zimmerman [1], elevan el número de ocurrencias de las instrucciones MOV, PUSH y POP.

Algunos autores han hecho ver que las dependencias de datos debidas a punteros a memoria son aún más baldías que las debidas a las transformaciones de datos residentes en registros [4, 2]. La idea es que las segundas derivan de la semántica del programa (lo que hace el algoritmo programado) mientras que las primeras son artificiales, debidas al modelo de programación, a la limitación en recursos físicos y además generan dependencias dobles: a través de los registros punteros y a través de la propia memoria considerada como recurso único. Hay toda una carga de programación, añadida a la tarea propia del algoritmo, que se encarga de ejecutar código para incrementar, decrementar y actualizar convenientemente los punteros de memoria.

b. Uso implícito de registros

La arquitectura del repertorio de instrucciones x86 trabaja con operandos implícitos asociados al código de operación que no aparecen, por tanto, en el formato. Es, igualmente, una manera de ahorrar espacio de representación. Que el programador (el compilador) no los exprese no significa que no generen dependencias de datos. Los operandos implícitos involucran, además, un uso dedicado de registros que puede agravar el problema de encontrar operaciones independientes para ejecutar en un entorno concurrente. También incrementa el número de transferencia con la pila.

En la figura 8 se presentan las gráficas que dan la distribución del uso implícito de registros para los distintos programas del banco de pruebas.

Se puede ver como los registros referenciados de modo implícito son siempre los mismos: acumulador (AX), contador (CX), SP, DI, SI y muy esporádicas ocurrencias sobre BX y DX.

El número de accesos implícitos supera en algunas ocasiones al de accesos explícitos. Tal es el caso de DEBUG, FIND y SORT.

Los accesos al puntero de pila (SP) vienen de la mano de instrucciones de manejo de la misma: PUSH, POP, CALL, RET. El trabajo de Postiff [4] identifica este hecho y analiza sus consecuencias.

En las gráficas no se han incluido los registros de segmento que aparecen siempre en el cálculo de las direcciones efectivas de memoria. Su número total es igual al de accesos a memoria y su distribución tiene que ver con el registro por defecto dado en la tabla 4 o los prefijos especificados, en su caso, de manera explícita con el fin de modificar el establecido por defecto. No obstante, en este trabajo no se ha contabilizado detalladamente cada ocurrencia.

En general, el número de lecturas y escrituras es prácticamente igual para todos los registros con la excepción del acumulador que suele tener menos escrituras. Este hecho contribuye a la aparición de largas cadenas de dependencias.

El resto de registros usados implícitamente en nuestras trazas se debe sobre todo a las operaciones con cadenas. LOOP también genera accesos implícitos al registro contador (CX) pero la cantidad atribuible a esta instrucción es muy pequeña ya que el uso que se hace de ella ni siquiera entra en el conjunto de las 25 más usadas en promedio.

Destaca la traza de SORT ya que el acceso a implícitos es muy superior al de explícitos (1.150.000 frente a 130.000 aproximadamente) debido al significativo uso de instrucciones de manejo de cadenas. El recurso a estos códigos de operación, que en realidad corresponden a primitivas de bucles, impide llevar a cabo optimizaciones con el fin de aprovechar posibles recursos paralelos.

c. Uso implícito de banderas de estado

Este repertorio de instrucciones es un claro representante de las arquitecturas basadas en registro de estado. En ellas, los saltos condicionales se evalúan en función del valor que contenga un registro especial constituido por una serie de campos que almacenan información sobre diferentes situaciones. Estos campos son actualizados por algunas de las instrucciones del repertorio de manera implícita e incondicional.

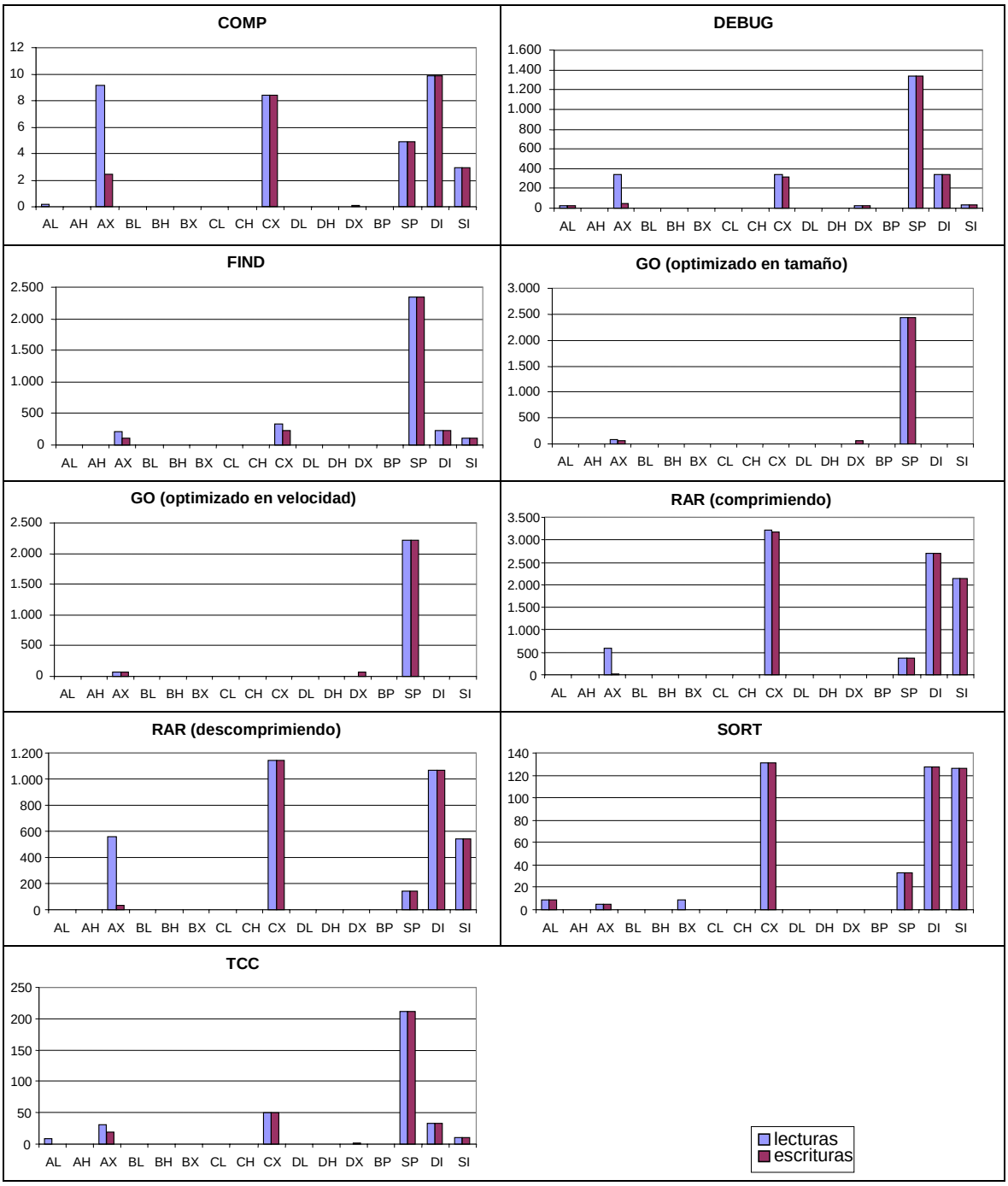

Fig. 8. Uso implícito de registros expresado en miles de referencias**.**

La figura 9 muestra la distribución del uso de banderas de estado por trazas. Se han señalado las lecturas y escrituras de los bits correspondientes a lo que *Intel* llama banderas de estado. El fabricante distingue en este registro entre banderas de estado, actualizadas por instrucciones de proceso, y banderas de control, gobernadas por el programador para establecer algún modo de funcionamiento. Nosotros estamos interesados en las primeras por depender directamente de la ejecución de determinados códigos de operación.

El acceso a las banderas de estado en escritura es especialmente importante en el caso de las instrucciones de proceso mientras que las lecturas suelen corresponder a las instrucciones de salto condicional. Algunas instrucciones de proceso leen las banderas como un dato más de entrada.

De las gráficas mostradas seguidamente se desprende que las escrituras se hacen en bloque mientras que las lecturas atienden a unos campos (bits) concretos. Por ejemplo, la traza de COMP sólo lee ZF y CF mientras que escribe en bloque en todas las banderas de estado (a excepción de CF); DEBUG, FIND, RAR y TCC se comportan de manera similar. Las dos trazas de GO también se ajustan a lo dicho: escriben en bloque pero sólo leen OF, SF y ZF.

Este modo de operar es totalmente explicable. La idea es que en cada bloque básico tenemos un salto.

Este salto es mayoritariamente un salto condicional que evalúa una condición expresada por una bandera de estado, a veces una combinación de dos de ellas y muy raras veces tres. En consecuencia, la ejecución de la bifurcación lee alguna bandera de estado concreta, no el bloque. Sin embargo, las instrucciones de proceso escriben el estado modificando la mayoría de los bits de estado.

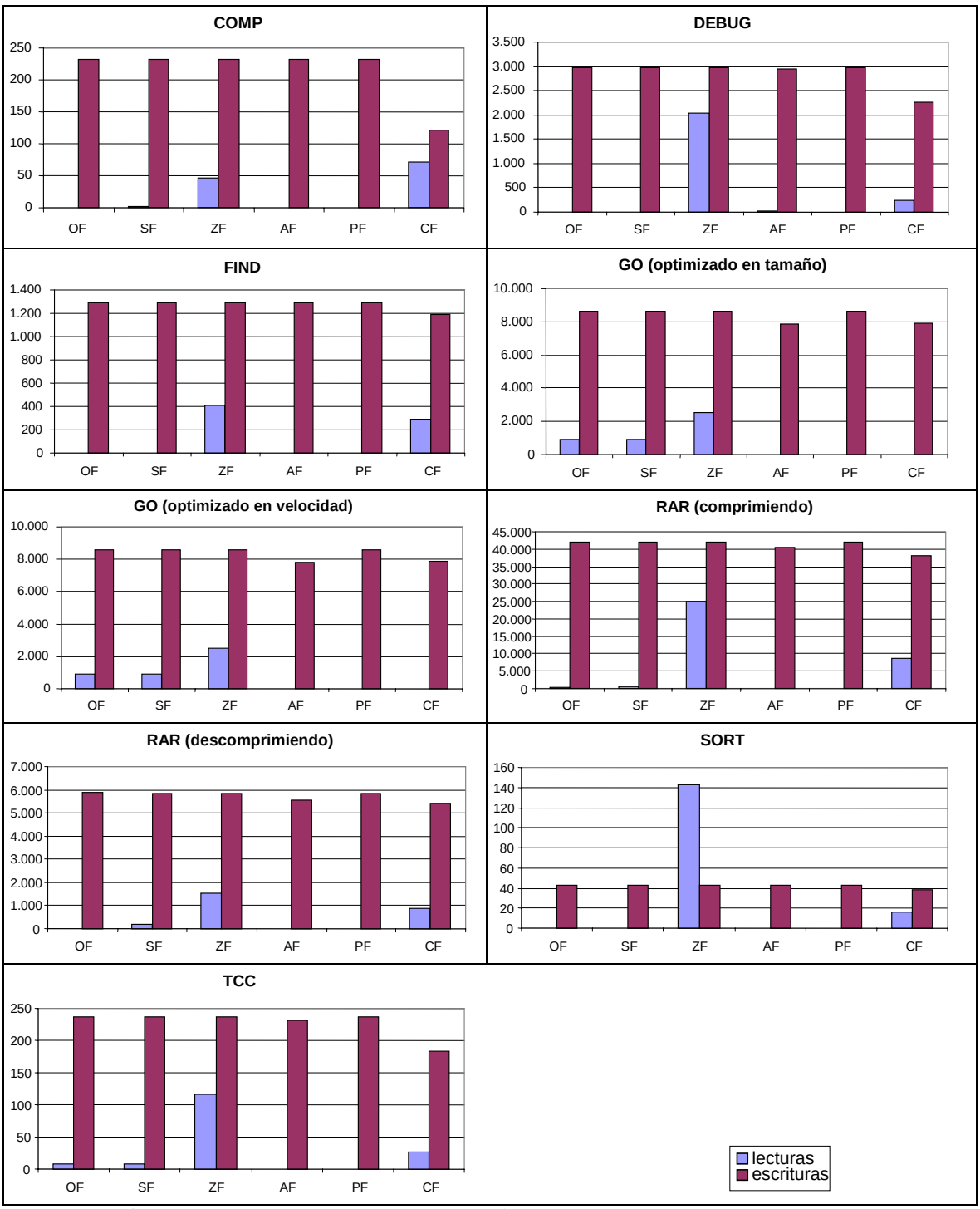

Fig. 9. Uso implícito de banderas de estado expresado en miles de referencias.

Hay que hacer notar otro hecho. El número de escrituras supera en la práctica totalidad de los casos a las lecturas. Tan sólo se sale de la regla la traza de SORT a consecuencia de un prefijo de repetición de instrucción de manejo de cadenas —nuevamente estas instrucciones sacan a este programa de la norma—.

Hay, por tanto, un desequilibrio entre la información generada y la información requerida tanto en extensión como en cantidad. En extensión porque escribo más datos acerca del estado de los que realmente necesito. En cantidad porque las escrituras de estado superan tres o cuatro veces a las lecturas.

¿A qué se debe que el número de escrituras supere al de lecturas? Volvamos al bloque básico. En cada uno de ellos tenemos, en promedio, más de una instrucción de proceso (de las que escriben el estado en bloque) por un solo salto condicional. El salto condicional tan sólo tiene en cuenta la última actualización de estado haciendo que las escrituras previas sean absolutamente inútiles. No solamente resultan improductivas sino que generan dependencias de datos. Si las dependencias de datos tratan cada bandera como un símbolo de dato, el grafo resultante tiene muchos arcos (dependencias de datos) y en consecuencia es "muy trabado".

Un número de escrituras muy grande sobre el mismo elemento conlleva un potencial incremento de las dependencias de salida. Es verdad que las dependencias de salida, así como las antidependencias, no son de las "verdaderas", las que tienen sentido computacional, pero el renombramiento para evitarlas supone tener un banco de registros de estado adicional.

Vamos a examinar nuestras trazas contando cuántas operaciones de proceso hay en promedio por bloque básico como modo de medir el grado de limitación al paralelismo impuesto por la arquitectura del repertorio de instrucciones y, por tanto, no atribuible a la semántica del programa. En la gráfica adjunta (Fig. 10) se puede ver que el número de operaciones de proceso y, por tanto, de escrituras en el registro de estado, por bloque básico supera al de lecturas, haciendo ver que muchas de ellas son superfluas. En la gráfica no se ha presentado SORT que, teniendo un porcentaje de instrucciones de manejo de cadenas altísimo, no puede tratarse, respecto al bloque básico, como al resto de trazas.

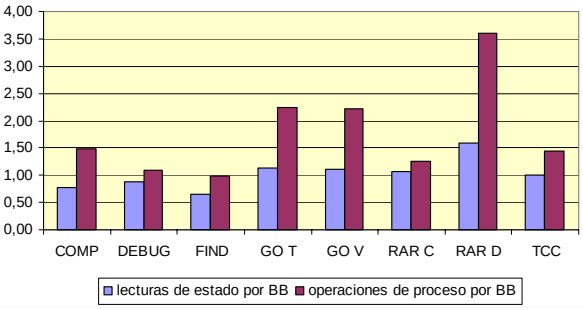

Fig. 10. Distribución de lecturas de estado y operaciones de proceso (escrituras de estado) por bloque básico.

Hay que explicar que no todas estas dependencias son superfluas ya que pueden superponerse a las reales (semánticas) generadas por el procesamiento de los datos. Sin embargo, si las obviamos podemos obtener el límite superior del paralelismo alcanzable. El grado de paralelismo real se moverá entre el límite actual y el superior calculado al evitar estas dependencias.

Lo visto hasta aquí tiene impacto en el grado de dependencias de datos de una manera directa. Ahora bien, el efecto completo puede minimizarse mediante técnicas de desambiguación: renombramiento de registros y de memoria fundamentalmente [7, 6, 2, 4]. La aplicación de estas técnicas es más sencilla en el caso de los registros del procesador que en el caso de la memoria. Asimismo, es más sencilla en el caso de los registros explícitos que en el caso de los implícitos y se hace muy complicada en el caso de los registros de estado en las arquitecturas basadas en estado, como la que nos ocupa.

4. 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] T. M. Austin and G. S. Sohi, "Dynamic Dependency Analysis of Ordinary Programs," in *Proceedings of the 19th International Symposium on Computer Architecture*, 1992, pp. 342-351.

[3] R. Hyde. *The Art of Assembly Language Programming*. Versión borrador. 2001. Available at: http://webster.cs.ucr.edu

[4] M. A. Postiff, *D.* A. Greene, G. S. Tyson and T. N. Mudge, "The Limits of Instruction Level Parallelism in SPEC95 Applications," in *Proceedings of the 3rd Workshop on Interaction Between Compilers and Computer Architecture*, 1998.

[5] R. Rico, "Proposal of test-bench for the x86 instruction set (16 bits subset)," *Technical Report TR-UAH-AUT-GAP-2005-21-en*, November 2005.
Available at: http://atc2.aut.uah.es/~gap/

[6] K. B. Theobald, G. R. Gao and L. J. Hendren, "On the Limits of Program Parallelism and its Smoothability," in *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pp. 10-19, 1992.

[7] D. W. Wall, "Limits of instruction-level paralelism," in *Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 176-188, April 1991.
Also as:
W.R.L. Research Report 93/06. Digital Equipment Corporation. Palo Alto, CA. 1993. Available at: http://www.research.compaq.com/wrl/techreports/index.html.