DANIEL MONJAS MIGUÉLEZ. DGIIM 2º.

TABLA 5.

1.Para N≈32, ¿cuántos bits adicionales pueden llegar a necesitarse para almacenar el resultado? Dicho resultado se alcanzaría cuando todos los elementos tomaran el valor máximo sin signo. ¿Cómo se escribe ese valor en hexadecimal? ¿Cuántos acarreos se producen? ¿Cuánto vale la suma (indicarla en hexadecimal)? Comprobarlo usando ddd.

Si hacemos la suma de 0xFFFFFFFF 32 veces nos da que el resultado será 0x1FFFFFFE0 y si representamos este numero en binario tenemos que se trata del numero 00011111111111111111111111111100000 que son 40 bits, pero sin embargo al tratarse de un entero sin signo se puede obviar los 0 de la izquierda luego se requerirían de 37 bits para representar el numero. Como estamos trabajando con numeros enteros de 32 bits sin signo se requieren 5 bits adicionales.
Para ver cuantos acarreos se producen nos fijamos en que el acarreo que da la operación de sumar 32 veces 0xFFFFFFFF es 0x0000001F (fijandose en el valor almacenado en $resultado+4 tras realizar la operación), si pasamos este numero a decimal tenemos que 0x0000001F=31 que es el numero de accareos producidos.
El valor de la suma ya la hemos especificado antes pero es 0x1FFFFFFE0.

2.Si nos proponemos obtener sólo 1 acarreo con una lista de 32 elementos iguales, el objetivo es que la suma alcance 2^32 (que ya no cabe en 32bits). Cada elemento debe valer por tanto 2^32/32 = 2^32/2? = ?. ¿Cómo se escribe ese valor en hexadecimal? Inicializar los 32 elementos de la lista con ese valor y comprobar cuándo se produce el acarreo.

Cada elementos debe valer 2^32/32=2^32/2^5=2^27=134217728(en decimal)=0x08000000(en hexadecimal)
El acarreo se produce en la iteración numero 32 del bucle suma, es decir, en la última suma del bucle.

3.Por probar valores intermedios: si la lista se inicializara con los valores 0x10000000, 0x20000000, 0x40000000, 0x80000000, repetidos cíclicamente, ¿qué valor tomaría la suma de los 32 elementos? ¿Cuándo se producirían los acarreos? Comprobarlo con ddd.

La suma de los valores de la lista toma el valor 0x780000000=32212254720 en decimal.
Los acarreos se producen en la iteraciones 5, 10, 14, 19, 23, 27, 31.

TABLA 6.
1.¿Cuál es el máximo entero positivo que puede representarse (escribirlo en hexadecimal)? Si se sumaran los N≈32 elementos de la lista inicializados a ese valor ¿qué resultado se obtendría (en hexadecimal)? ¿Qué valor aproximado tienen el elemento y la suma (indicarlo en múltiplos de potencias de 10)? Comprobarlo usando ddd.

Como tratamos con numeros de 32 bits con signo debemos reservar el primer bit para indicar el signo y el resto de bits para la representación del numero, luego el maximo número que se puede sumar sería
01111111111111111111111111111111=0x7FFFFFFF (binario|hexadecimal).
El resultado sería 0x0FFFFFFFe0 en hexadecimal

Los valores aproximados del valor en decimal serían:
valor:2*10^9
suma:7*10^10

2.Misma pregunta respecto a negativos: menor valor negativo en hexadecimal, suma, valores decimales aprox., usar ddd.

Razonando de la misma manera disponemos de numeros representados con 32 bits, de los cuales el primero se reserva para el signo. El menos numero decimal que se puede escribir con 32 bits teniendo en cuenta el signo estaría compuesto por 32 unos y su representación en hexadecimal sería 0xFFFFFFFF.
La suma de todos ellos da lugar al mismo numero que la suma del ejercicio anterior cambiando el signo, es decir, 0x1FFFFFFFe0.
Los valores aproximados del valor en decimal serían:
valor:-2*10^9
suma:-7*10^10

3.Si nos proponemos obtener sólo 1 acarreo con una lista de 32 elementos positivos iguales, se podría pensar que el objetivo es que la suma alcance 2^31 (que ya no cabe en 32bits como número positivo en complemento a dos). Aparentemente, cada elemento debe valer por tanto 2^31/32 = 2^31/2^5 = ?. ¿Cómo se escribe ese valor en hexadecimal? Inicializar los 32 elementos de la lista con ese valor y comprobar si se produce el acarreo.

Cada elemento debe valer 2^26=0x04000000 es la representación del valor en hexadecimal.
Si escribimos la lista con 32 veces el valor escrito el resultado que da es 0x80000000 en hexadecimal, que es igual 2^31, y además no se produce acarreo, pero esto se debe a que para la representación del resultado usamos un quad y no un int, por lo que disponemos de 64 bits, luego no se produce acarreo puesto que el programa esta hecho de tal forma para que los 32 bits destinados al acarreo primero contienen el signo del numero y posteriormente se va aumentando el valor de estos bits para ir almacenando el acarreo que se produce solo si una suma excede los 32 bits. Luego realmente el resultado que tenemos es
0x0000000080000000 que es 2^31 con signo positivo.

4.Repetir el ejercicio anterior de forma que sí se produzca acarreo desde los 32bits inferiores a los superiores. ¿Cuál es el valor de elemento requerido? ¿Por qué es incorrecto el razonamiento anterior? Indicar los valores decimales aproximados (múltiplos de potencias de 10) del elemento y de la suma. Comprobarlo usando ddd.

Por lo explicado en en el anterior apartado el resultado de la suma se almacena en 32 bits + 32bits que contienen acarreo y signo, luego el acarreo se produce si se supera 2^32 es decir si se suman 32 veces 2^27.
El razonamiento anterior es incorrecto porque no se tiene en cuenta que en cada operacion se extiende el signo del valor que se va a sumar de forma que el numero se representa EDX:EAX donde a EDX se ha extendido el signo del numero que se va a sumar. El adc que se realiza un poco mas abajo tiene esto en cuenta y suma la extensión del signo con el acumulador del acarreo. Es decir que realmente trabajamos con 64 bit, 32 para el signo y el acarreo y otros 32 para representar el numero que se suma, es por eso que el razonamiento anterior no valía, porque operaba solo sobre 31 bits para reservar siempre uno para el signo, y pensaba que al utilizar el bit del signo para representar el numero se produciría acarreo, pero no es asi.

Los valores aproximados en potencias de 10 son:
valor:1*10^8
suma:4*10^9

5. Respecto a negativos, 2^31 sí cabe en 32bits como número negativo en complemento a dos. Calcular qué valor de elemento se requiere para obtener como suma 2^31, y para obtener 2^32. Comprobarlo usando ddd.

-2^31/32=-2^31/2^5=-2^26 que si lo expresamos en hexadecimal es 0x84000000 (Funciona comprobado con ddd).
-2^32/32=-2^32/2^5=-2^27 que si lo expresamos en hexadecimal es 0x88000000 (Funciona comprobado con ddd).

6. Por probar valores intermedios: si la lista se inicializara con los valores 0xF0000000, 0xE0000000, 0xE0000000, 0xD0000000, repetidos cíclicamente, ¿qué valor tomaría la suma de los 32 elementos (en hex)? Comprobarlo con ddd.

El valor que toma la suma de los numeros en hexadecimal es 0xFFFFFFFC00000000.

TABLA 7.
1.Rellenando la lista al valor -1, la media es -1. Cambiando un elemento a 0, la media pasa a valer 0. ¿Por qué? Consultar el manual de Intel sobre la instrucción de división. ¿Cuánto vale el reto de la división en ambos casos?. Probar con ddd.

Se debe a que la suma pasaría a valer -31 y por tanto la división de -31/32 almacena sólo la parte entera de esta división en la variable media. Según en manual para operandos de tamaños doble palabra como es nuestro caso (EDX:EAX) se almacena en EAX el cociente y el resto en EDX. La parte entera de esta división es cero, por consiguiente es lo que se almacenará en media. El resto de la división en el caso de que todos los valores sean -1 es 0 mientras que en el caso de que sean todos los valores -1 menos un 0 da como resto 0xFFFFFFE1.

2.También se obtiene 0 si se cambia lista[0]=1, 2, 3… Para facilitar el cálculo mental, podemos ajustar lista[1]=-2, y así la suma lista[0]=-32, resultando más facil calcular el resto. ¿Para qué rango de valores de lista[0] se obtiene cociente 0? ¿Cuánto vale el resto a lo largo de ese rango? Comprobar que coinciden los signos de la suma del dividendo(suma) y del resto. NOTA: Para evitar el ciclo editar ensamblar enlazar depurar, se pueden poner un par de breakpoints antes y después de llamar la subrutina que calcula la media. Tras encontrar el primer breakpoint se puede modificar lista[0] con el comando set var lista=<valor>. Pulsar Cont para llegar al segundo breakpoint y ver en EAX el resultado retornado por la subrutina. Dependiendo de si se restauran los valores de los otros registros antes de retornar, es posible que sea ventajoso poner el segundo breakpoint antes de POP EDX para ver el valor del resto al mismo tiempo que el cociente. Para hacer muchas ejecuciones seguidas, puede merecer la pena (re)utilizar la línea de comandos (run/set var…/cont) en lugar del ratón.

Para que el cociente sea 0 se debe dar como condición que el dividendo, es decir, la suma de los valores este comprendida entre -32<suma<32. Como desde lista[1], hasta lista[31] todos los valores son -1 sabemos que la suma desde lista[1] hasta lista[31] es -31 luego el menor valor que puede

tomar lista[0] para que el cociente sea 0 es 0, y para que sea menor que 32 tenemos que lista[1]+(-31)<32, entonces lista[1]<63 luego lista[0] puede tomar cualquier valor entero entre 0 y 62 ambos incluidos (siempre que el resto de valores de la lista sean -1). Para ver entre que rango de valores se establece el resto calcularemos el valor del resto para 0 y el valor del resto para 62 y sabemos que todos los demas restos estarán comprendidos entre ellos. El primero ya le conocemos que es 0xFFFFFFE1 por el ejercicio anterior. El segundo lo hayamos usando ddd y es 0x0000001F.
Luego el resto toma valores en el intervalo [0xFFFFFFE1-0x0000001F].
Como se puede comprobar en ambos casos coincide el signo del cociente y el dividendo, en el primero es -31/32 y el resto es 0xFFFFFFE1 cuyo primer bit, el del signo, es 1 por lo que es un numero negativo. En el segundo caso el dividendo es 31 y el resto 0x0000001F cuyo primer bit es 0 luego es un número positivo.

3.¿Para qué rango de valores de lista[0] se obtiene media 1? ¿Cuánto vale el resto en ese rango? Comprobarlo con ddd, y notar que tanto las sumas como los restos son positivos (el cociente se redondea hacia cero).

Haremos el mismo razonamiento que en caso anterior. Si todos los valores de la lista menos el primero son menos uno el valor de la suma será lista[0]-31 y para que este valor dividido entre cualquier otro de media 1 se deve verificar que la parte entera de la división sea 1. Para que la parte entera de la división sea 1 se tiene que dar que lista[0]-31>=32, en otro caso el cociente sería 0 (no incluyo que la suma sea <=-32 porque entonces el cociente sería -1 y no 1.). Despejando la inecuación obtenemos que lista[0] tiene que ser al menos mayor o igual que 63. Ahora bien para que el cociente no sea mayor que uno lista[0]-31<2*32, entonces lista[0]<95. Luego se tomará valores de lista[0] entre 63<=lista[0]<=95. Y para el resto se tomará el rango comprendido entre el resto para lista[0]=63 y el resto para lista[0]=95.

Como ya veníamos diciendo para lista[0]=63 se tiene que el resto es 0x00000000 y el cociente 1 y para lista[0]=94 se tiene que el cociente es 1 y el resto es 0x0000001F, luego al igual que en el caso anterior se toman valores entre [0x00000000-0x0000001F]. El intervalo de restos se repetirá siempre el mismo para cualquier cociente que se nos pida hallar ya que el numero de lista[0] existientes para un valor del cociente es siempre el mismo y la distancia entre ellos es siempre la misma. Se puede ver que el valor del signo del resto y el del dividendo coincide.

4. ¿Para qué rango de valores de lista[0] se obtiene media -1? ¿Cuánto vale el resto en ese rango? Comprobarlo con ddd, y notar que tanto las sumas como los restos son positivos (el cociente se redondea hacia cero).

Razonando de forma análoga a la del anterior ejercicio llegamos a que lista[0]-31<=-32, entonces lista[0]<=-1. También hay que resaltar que lista[0]-31<2*-32, entonces lista[0]<-33, luego el rango de valores de lista[0] es -33<lista[0]<=-1. Utilizamos ddd para comprobar y para calcular el rango de los restos que esta entre el resto para lista[0]=-1 y lista[0]=-32. Ya sabemos que para lista[0]=-1 todos los valores de lista son -1 y por tanto el resto es 0, mientras que para lista[0]=-32 el resto es 0xFFFFFFE1 y el valor del cociente es -1, por lo que el rango de valores para el resto es [0x00000000-0xFFFFFFE1]. Finalmente, se puede ver que el valor del resto es igual que el del dividendo con la única excepción de cuando el 0 es el resto pues este no tiene signo (es decir, que no tiene el bit del signo a 1 por ser el 0).