El formato BCD o *Binary Coded Decimal* se usa para representar números enteros positivos en base 10. Cada cifra decimal del número se codifica en binario en 4 bits. Por ejemplo el número 60219 se codifica como 0b0110 0000 0010 1001, en donde el 0110 codifica el 6, el 0000 el 0, el 0010 el 2 y el 1001 el 9. La ventaja es que al anotar la secuencia de bits en hexadecimal se lee exactamente el mismo número representado en BCD, es decir 0x60219. Observe que las secuencias 0b1010 … 0b1111 o 0xa … 0xf no pueden ocurrir y que 0x60219 representa en BCD el número 60219 mientras que el mismo 0x60219 representa en binario el número 393753 ($9 + 1*16+2*16^2+6*16^4$).

Programe la función *sumaBcd* que retorna la suma de 2 números en formato BCD, almacenados en enteros sin signo de 64 bits, es decir *unsigned long long*. El número más grande representable puede tener hasta 16 cifras decimales. Si el resultado de la suma tiene 17 cifras decimales se produce desborde y Ud. debe retornar 0xffffffffffffffff. El encabezado de la función *sumaBcd* es:

```
typedef unsigned long long Bcd;
Bcd sumaBcd(Bcd x, Bcd y);
```

Ejemplo de uso:

```
Bcd a= sumaBcd(0x60219, 0x1); // a es 0x60220
Bcd b= sumaBcd(0x199305, 0x9781); // b es 0x209086
Bcd c= sumaBcd(0x9999999999999999, 0x1); // c es 0xfff...ffff
```

Observe que no sirve sumar directamente los números en BCD con el operador + de C porque 0x60219 + 0x1 es 0x6021A, que es una secuencia de bits inválida en BCD. Por lo tanto Ud. necesita separar las cifras decimales de a 4 bits y sumar exactamente como aprendió a hacerlo en enseñanza básica.

Restricciones:

- Ud. no puede usar los operadores de multiplicación, división o módulo (* / %). Use los operadores de bits eficientemente.
- No se permite convertir los números a binario, sumarlos con + y convertir el resultado a BCD.
- Se descontará medio punto por no usar el estilo de indentación de Kernighan como se explica en esta sección de los apuntes.
- El estándar de C no especifica el resultado para desplazamientos mayores o iguales al tamaño del operando. Sanitize rechaza el desplazamiento *x*<<*nbits* cuando *nbits* es mayor o superior a la cantidad de bits de *x*.

Instrucciones

Baje *t1.zip* de U-cursos y descomprímalo. El directorio *T1* contiene los archivos (a) *test-suma.c* que prueba si su tarea funciona y compara su eficiencia con la solución del profesor, (b) *prof.ref-x86_64 y prof.ref-aarch64* con los binarios ejecutables de la solución del profesor, (c) *suma.h* que incluye el encabezado de la función pedida, y (d) *Makefile* que le servirá para compilar y ejecutar su tarea. Ud. debe programar la función *sumaBcd* en el archivo *suma.c*.

Pruebe su tarea bajo Debian 11 de 64 bits nativo o virtualizado con VirtualBox, Vmware, QEmu o WSL 2. **Ejecute el comando *make* sin parámetros**. Le mostrará las opciones que tiene para compilar su tarea. Estos son los requerimientos para aprobar su tarea:

- *make run* debe felicitarlo por aprobar este modo de ejecución. Su solución no debe ser 80% más lenta que la solución del profesor.
- *make run-g* debe felicitarlo.
- *make run-san* debe felicitarlo y no reportar ningún problema como por ejemplo desplazamientos indefinidos.

Cuando pruebe su tarea con *make run* asegúrese que su computador esté configurado en modo alto rendimiento y que no estén corriendo otros procesos intensivos en uso de CPU al mismo tiempo. De otro modo podría no lograr la eficiencia solicitada.

Entrega

Ud. solo debe entregar por medio de U-cursos el archivo *suma.zip* generado por el comando *make zip*. **A continuación es muy importante que descargue de U-cursos el mismo archivo que subió, luego descargue nuevamente los archivos adjuntos y vuelva a probar la tarea tal cual como la entregó**. Esto es para evitar que Ud. reciba un 1.0 en su tarea porque entregó los archivos equivocados. Créame, sucede a menudo por ahorrarse esta verificación. Se descontará medio punto por día de atraso. No se consideran los días de receso, sábados, domingos o festivos.