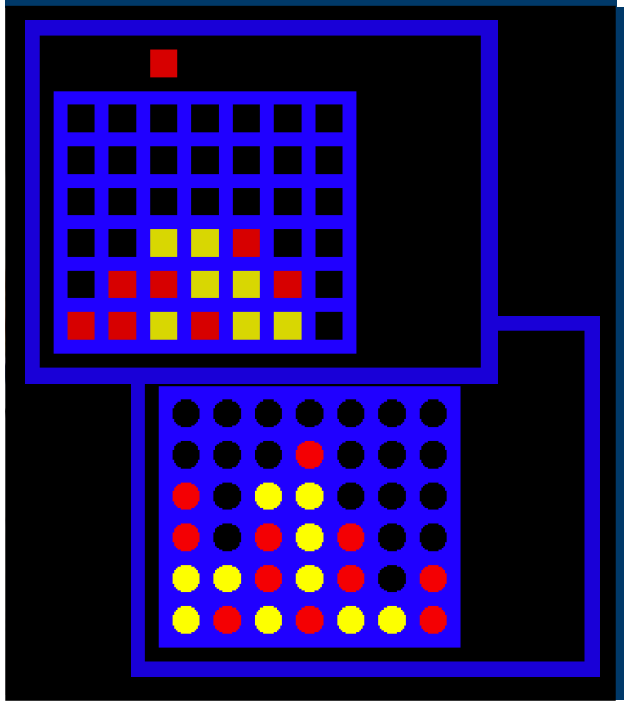
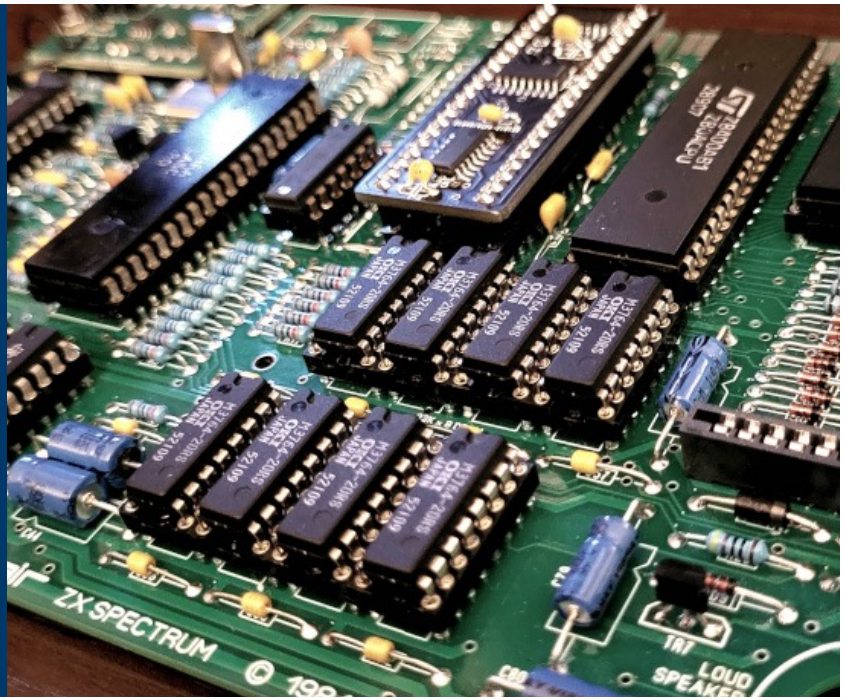
Arquitectura y Organización de Computadores

CONECTA-4 Z80 ASM HACKATHON





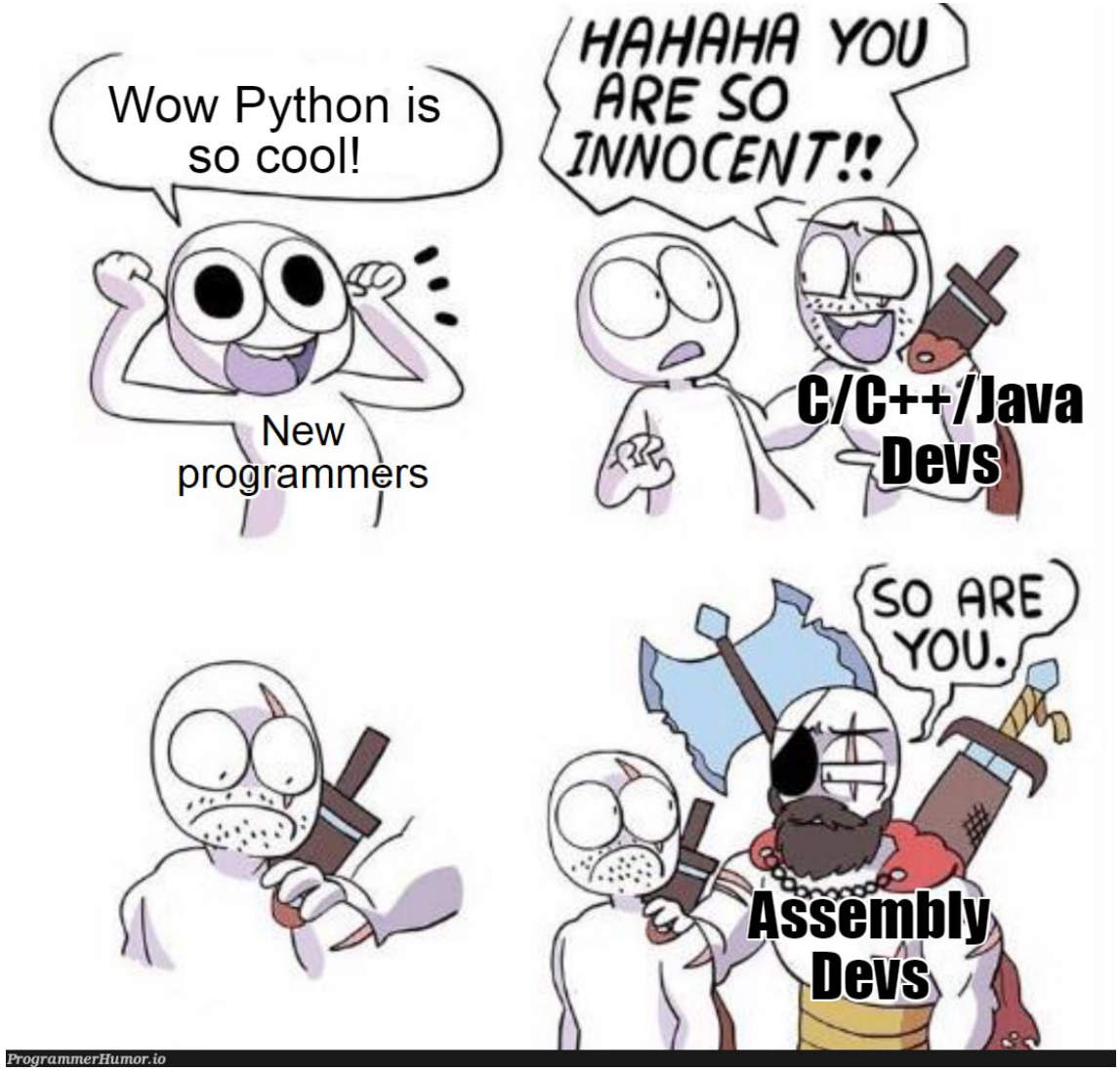
2025-26 daniel.leon@ufv.es

ENSAMBLADOR…

"Dice aquí que deberías estar en el infierno, pero como has programado en ensamblador, lo daremos como sentencia ya cumplida"

Es complejo, pero….

ENSAMBLADOR...



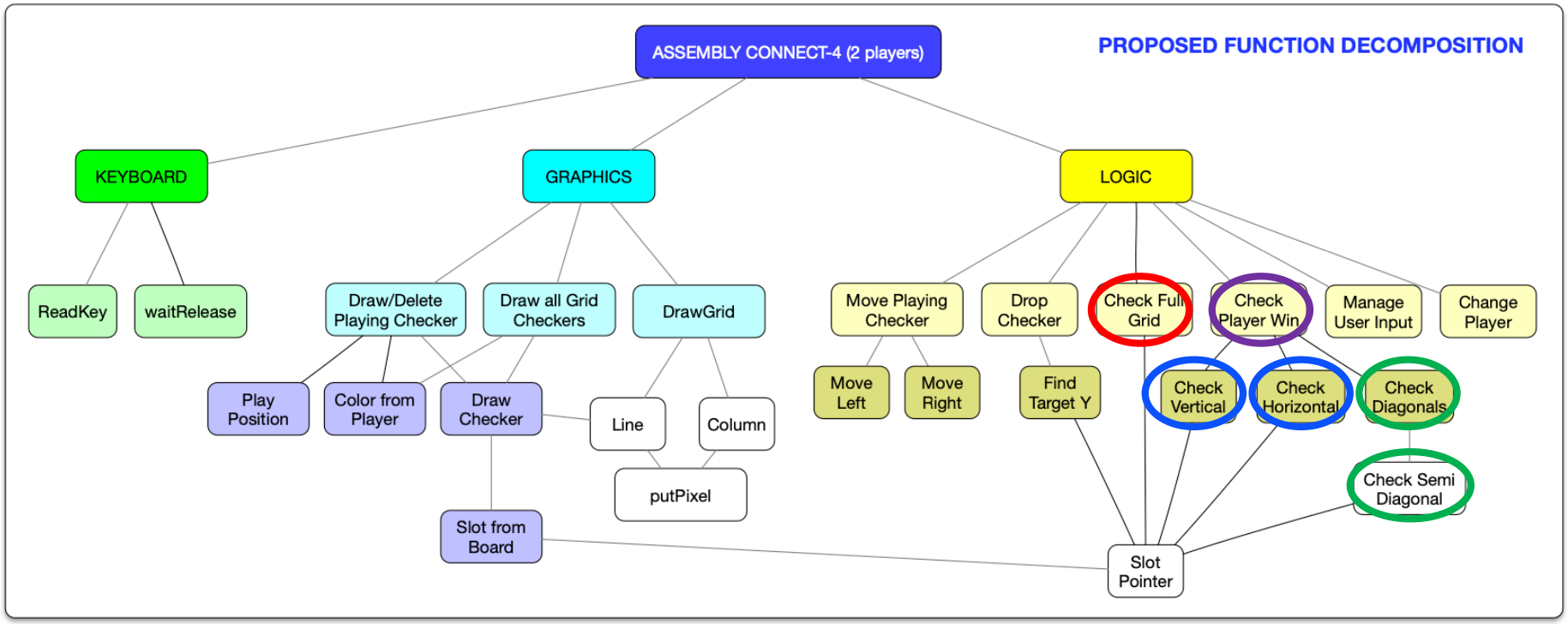
"Guau, Python es tan chulo"

"jajaja, eres tan inocente"

"y tú también"

… Te hace mejorar.

RETOS



1. Detectar fin de juego por empate: Tablero lleno
2. Detectar ganador
    1. 4-en-raya vertical
    2. 4-en-raya horizontal
    3. 4-en-raya diagonal
        - Comprobar semidiagonales

Consideraciones previas (I)

- Asegúrate de tener lo siguiente:
  - Estructura de datos del juego: Array X*Y
  - Función ***slotPointer*** desarrollada y verificada
  - Capacidad de volcar contenidos de la estructura de datos a pantalla

- Opcionalmente, y para un mayor aprovechamiento:
  - Jugadores se alternan para soltar piezas al tablero
  - Las funciones de teclado están implementadas
  - La función ***dropChecker*** está implementada

En este punto, el juego es un *simulador* de un tablero físico, pero no realiza ninguna comprobación

Consideraciones previas (II)

- Es conveniente tener las variables del juego en una estructura para acceder con ellas en modo Base+Desplazamiento

```
GAMESTRUCT:                     ; Game structure for indirect+offset addressing
Level:              DEFB 0      ; Current level
CurrentPlayer:      DEFB 1      ; Current player (0=None, 1=Player1, 2=Player2)
CurrentSlot:        DEFB 0      ; Current slot selected (0 to BoardXSize-1)
PointsP1:           DEFB 0      ; Player 1 points
PointsP2:           DEFB 0      ; Player 2 points
SlotColors          DEFB BACKGROUNDCOLOR, P1COLOR, P2COLOR
PlayerChangeP:      DEFB 0      ; Player Change Pending if <> 0
LastY:              DEFB $FF    ; Last inserted Y coordinate
pieceCount          DEFB 0      ; Count of consecutive pieces
Board:              DS XSIZE*YSIZE, 0 ; Board slots (0=empty, 1=Player1, 2=Player2)

LEVELoff:           EQU Level-GAMESTRUCT            ; Level counter offset
CURRENTPoff:        EQU CurrentPlayer-GAMESTRUCT    ; Current player offset
CURRENTSoff:        EQU CurrentSlot-GAMESTRUCT      ; Current slot offset
POINTSP1off:        EQU PointsP1-GAMESTRUCT         ; Player 1 points offset
POINTSP2off:        EQU PointsP2-GAMESTRUCT         ; Player 2 points offset
SLOTCOLoff:         EQU SlotColors-GAMESTRUCT       ; Slot Colors 3 bytes
CPPENDINGoff:       EQU PlayerChangeP-GAMESTRUCT    ; PlayerChange Pending offset
LASTYoff            EQU LastY-GAMESTRUCT            ; Last inserted Y offset
COUNToff            EQU pieceCount-GAMESTRUCT       ; Count for consecutive pieces in diagonal
BOARDoff:           EQU Board-GAMESTRUCT            ; Board offset (start of board array)
```

Nota: Vuestras variables pueden ser distintas

Si el juego mantiene en el **registro IY (o el registro IX)** la dirección de memoria de *GAMESTRUCT* (que es la misma que la de *Level* en este ejemplo), se puede acceder a cualquier variable con Base+Desplazamiento.

Ejemplos:

```
LD C,  (IY+LASTYoff)

LD (IY+CURRENTPoff), E

LD B, (IY+COUNToff)
```

Detectar tablero lleno: Función

- **Recibe:**
  - Nada: Tiene acceso al array del tablero
- **Devuelve:**
  - A=0 si el tablero no está lleno
  - A=cualquier valor distinto de 0 (en la práctica 1 o 2) si está lleno
- **Usa:**
  - Podría usar la Función slotPointer, que recibe una coordenada X,Y **del tablero** (**no de la pantalla)** y devuelve en HL el puntero dentro del array a la posición donde está guardado lo que hay en ese slot. Pero como veremos, no es necesario.
- **Notas:**
  - Esta función debe ser llamada cada vez que se inserta una ficha nueva, pero sólo tras comprobar primero si ha ganado algún jugador (siguiente reto).
  - Es un bucle que recorre todas las X del tablero en la fila 0, la más alta.
  - Como esta fila está en las primeras posiciones del array, realmente se trata de mirar las XSIZE (tamaño X del tablero) primeras posiciones, y salir del bucle si se encuentra un espacio vacío.
  - Recordad que el tablero contiene:
    - 0: Espacio vacío
    - 1: Pieza de jugador 1
    - 2: Pieza de jugador 2

Detectar tablero lleno: Función (II)

```
uint8_t checkFullGrid(uint8_t* array) {
    uint8_t xLoop = 0;
    uint8_t slotValue = 0xFF;

    while ((xLoop<XSIZE) && (slotValue!=0)) {
        slotValue=array[xLoop];
        xLoop++;
    }
    return slotValue;
}
```

* Este programa no es necesariamente la mejor solución en C, pero tiene ciertas similitudes con lo que tendrás que hacer en ensamblador.

Pistas:

- Debido a las reglas del juego, basta con comprobar la fila superior. Si está llena es porque todas las inferiores están llenas también.
- Tal y como se configura el array del tablero de juego, la fila superior está en las primeras XSIZE posiciones.

N en raya horizontal: Función

- **Recibe:**
  - Posición Y en la que se ha colocado la última pieza.
  - Jugador actual (1 o 2)
  - Tiene acceso al array del tablero y a un define con el número de piezas necesarias para ganar (por defecto 4)
- **Devuelve:**
  - A=0 si hay N en raya
  - A=-1 si no hay N en raya
- **Usa:**
  - Usa la Función slotPointer, que recibe una coordenada X,Y **del tablero** (**no de la pantalla)** y devuelve en HL el puntero dentro del array a la posición donde está guardado lo que hay en ese slot.
- **Notas:**
  - Esta función debe ser llamada cada vez que se inserta una ficha nueva, dentro de las comprobaciones que hay que hacer para saber si el jugador ha ganado).
  - Veremos que hay una forma general de hacerla más adelante, pero esta función se puede realizar, simplemente, recorriendo todas las X del tablero para la Y donde se ha depositado la última y contar las sucesivas

N en raya horizontal: Función (II)

```
uint8_t checkHorizontalN (uint8_t* array, uint8_t yDropPosition, uint8_t player) {
    uint8_t consecutiveCheckers = 0;
    uint8_t xLoop = 0;

    while ((xLoop<XSIZE) && (consecutiveCheckers<WINSIZE)) {
        if (slotPointer(xLoop, yDropPosition, array) == player)
            consecutiveCheckers++;
        else
            consecutiveCheckers=0;
        xLoop++;
    }
    return (consecutiveCheckers == WINSIZE? 0 : -1);
}
```

Pistas:

- Es mejor comprobar desde XSIZE-1 hasta 0, incluyendo el 0.
- Para incluir el 0, es conveniente hacer el bucle con el flag de SIGNO (s), mediante:
  - JP P, label: Salta al label si el signo es positivo
  - JP M, label: Salta al label si el signo es negativo
- Ten en cuenta que el flag de signo sólo se puede usar con JP (salto absoluto), no con JR (salto relativo).
- Una opción es simplificar el bucle y sólo salir al final de la comprobación de la fila. En este caso, hay que comprobar si es >= WINSIZE.

* Este programa no es necesariamente la mejor solución en C, pero tiene ciertas similitudes con lo que tendrás que hacer en ensamblador.

N en raya vertical: Función

- **Recibe:**
  - Posición X en la que se ha colocado la última pieza.
  - Posición Y en la que se ha colocado la última pieza.
  - Jugador actual (1 o 2)
  - Tiene acceso al array del tablero y a un define con el número de piezas necesarias para ganar (por defecto 4)
- **Devuelve:**
  - A=0 si hay N en raya
  - A=-1 si no hay N en raya
- **Usa:**
  - Usa la Función slotPointer, que recibe una coordenada X,Y **del tablero** (**no de la pantalla)** y devuelve en HL el puntero dentro del array a la posición donde está guardado lo que hay en ese slot.
- **Notas:**
  - Esta función debe ser llamada cada vez que se inserta una ficha nueva, dentro de las comprobaciones que hay que hacer para saber si el jugador ha ganado).
  - Veremos que hay una forma general de hacerla más adelante, pero esta función es una mejora sobre la función horizontal, aprovechando el hecho de que todas las piezas están verticalmente agrupadas en Y (Desde el fondo hasta una altura), por tanto, la Y de más arriba será la de la pieza que acabamos de insertar

N en raya vertical: Función (II)

```c
uint8_t checkVerticalN (uint8_t* array, int8_t yDropPosition, int8_t xDropPosition, uint8_t player) {
    uint8_t consecutiveCheckers = 0;

    while ((yDropPosition<YSIZE) && (consecutiveCheckers<WINSIZE)) {
        if (slotPointer(xDropPosition, yDropPosition, array) == player)
            consecutiveCheckers++;
        else
            consecutiveCheckers=0;
        yDropPosition++;
    }
    return (consecutiveCheckers == WINSIZE? 0 : -1);
}
```

Pistas:

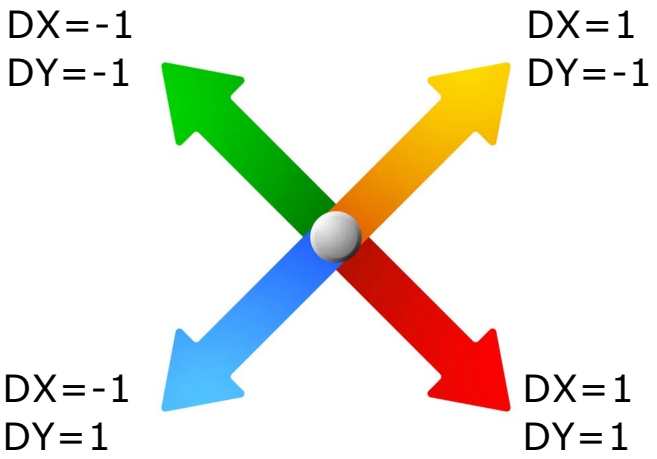
- Además de las optimizaciones de N en raya Horizontal, aquí hay una adicional: si (YSIZE – yDropPosition) < WINSIZE, es seguro que no podemos obtener las N en raya y se puede salir directamente de la función con valor -1

* Este programa no es necesariamente la mejor solución en C, pero tiene ciertas similitudes con lo que tendrás que hacer en ensamblador.

Semidiagonales

- Para calcular las N-en-raya en diagonal partiremos siempre de la posición X, Y de la pieza recién insertada
- Desde esta posición es posible tener N-en-raya en dos diagonales distintas, pero ¿dónde empezamos a mirar?
- La mejor opción es dividir cada diagonal en dos semidiagonales. Una hacia un lado de la pieza recién insertada y otra hacia el otro.
- Te tal forma que moviéndonos con DeltaX y DeltaY, podremos recorrer todas las semidiagonales (dos en cada diagonal):

  - Diagonal 1: DX,DY = (-1,-1)(1, 1)
  - Diagonal 2: DX,DY = (-1, 1)(1, -1)



Comprobar diagonal: Algoritmo alto nivel

- La función checkSemiDiagonal devolverá cuantas piezas del jugador consecutivas hay desde el punto de inserción hasta encontrar una pieza distinta, un hueco vacío o el final del tablero.

- Comprobar una diagonal supone:
  - Decidir qué diagonal de las dos posibles queremos comprobar:
  - Llamar a la función checkSemiDiagonal con los DX, DY adecuados para la mitad de esa diagonal, guardar el número devuelto (SD1N).
  - Llamar a la función checkSemiDiagonal con los otros dos DX,DY. Responderá con otro número (SD2N)
  - Calcular Consecutive=SD1N+SD2N+1 (el 1 es la pieza recién insertada)
  - Si consecutive >= WINSIZE (N para ganar), hay N en raya.

checkSemiDiagonal: Función

- **Recibe:**
  - Posición X en la que se ha colocado la última pieza.
  - Posición Y en la que se ha colocado la última pieza.
  - Jugador actual (1 o 2)
  - DeltaX y DeltaY para avanzar, según lo explicado anteriormente
  - Tiene acceso al array del tablero
- **Devuelve:**
  - Número de piezas consecutivas del color del jugador actual, desde la posición de inserción – Sin contar con la pieza recién insertada.
- **Usa:**
  - Usa la Función slotPointer, que recibe una coordenada X,Y **del tablero** (**no de la pantalla)** y devuelve en HL el puntero dentro del array a la posición donde está guardado lo que hay en ese slot.
- **Notas:**
  - Esta función debe ser llamada desde una función general de ”checkDiagonal”, que llama dos veces, por cada diagonal, a esta función y suma los resultados, según lo explicado en ”Algoritmo de alto nivel” en la diapositiva anterior.

checkSemiDiagonal: Función (II)

```
uint8_t checkSemiDiagonal(uint8_t* array, int8_t yDropPosition, int8_t xDropPosition, uint8_t player,
                          int8_t DX, int8_t DY ) {
    bool sigueIterando=true;
    uint8_t consecutiveCheckers = 0;

    while (sigueIterando) {
        yDropPosition+=DY;
        xDropPosition+=DX;
        sigueIterando =    (yDropPosition>-1) && (yDropPosition<YSIZE)
                        && (xDropPosition>-1) && (xDropPosition<XSIZE);
        if (sigueIterando) {
            if (slotPointer(xDropPosition, yDropPosition, array) == player)
            consecutiveCheckers++;
        else
            sigueIterando=false;
        }
    }
    return consecutiveCheckers;
}
```

Pistas:

- En ensamblador, los “and” (&&) de comprobaciones pueden hacerse de manera inversa, con comprobaciones sucesivas y saltos a la salida del bucle, esto es, cuatro comprobaciones de igualdad sobre -1 y XSIZE, YSIZE y cuatro saltos, uno tras cada comprobación
- Lo mismo con la comprobación del color del jugador (rama ”else”)

* Este programa no es necesariamente la mejor solución en C, pero tiene ciertas similitudes con lo que tendrás que hacer en ensamblador.

checkDiagonal: Función

- **Recibe:**
  - Posición X en la que se ha colocado la última pieza.
  - Posición Y en la que se ha colocado la última pieza.
  - Jugador actual (1 o 2)
  - DX, DY de la primera semidiagonal
- **Devuelve:**
  - Número de piezas consecutivas del color del jugador actual en la diagonal seleccionada (mediante el primer par DX, DY)
- **Usa:**
  - Usa la función checkSemiDiagonal.
- **Notas:**
  - Esta función debe ser llamada desde la función checkWIN
  - Es posible hacer una función genérica para comprobar las dos diagonales "checkDiagonals", tal y como está en el diagrama de descomposición, pero entonces no es tan fácil hacer una función general de comprobación, como comentaremos más adelante.

checkDiagonal: Función (II)

```
uint8_t checkDiagonal(uint8_t* array, int8_t yDropPosition, int8_t xDropPosition, uint8_t player,
                      int8_t DX, int8_t DY ) {
    uint8_t SD1N;
    uint8_t SD2N;

    SD1N = checkSemiDiagonal(array, yDropPosition, xDropPosition, player, DX, DY);
    SD2N = checkSemiDiagonal(array, yDropPosition, xDropPosition, player, DX*-1, DY*-1);
    return SD1N+SD2N+1;
}
```

Pistas:

- La multiplicación por -1 de un valor de 8-bit, es decir, su complemento a 2, se puede hacer fácilmente con la instrucción Z80 “NEG”, que realiza A=-A (A=A*-1)

* Este programa no es necesariamente la mejor solución en C, pero tiene ciertas similitudes con lo que tendrás que hacer en ensamblador.

Comprobar diagonal: Notas finales

- Ahora sólo faltaría una función superior “checkDiagonals” que llamara a checkDiagonal con DX=1, DY=1 y, si el resultado devuelto es menor que WINSIZE, llamarla otra vez con DX=1, DY=-1.
- Si en cualquiera de las dos llamadas, el resultado devuelto es mayor o igual a WINSIZE, el jugador tiene N-en-raya
- Esta función, por similitud a las de comprobación horizontal/vertical, debería devolver 0 en el caso de N-en-raya y -1 en caso contrario

checkWIN: Notas finales

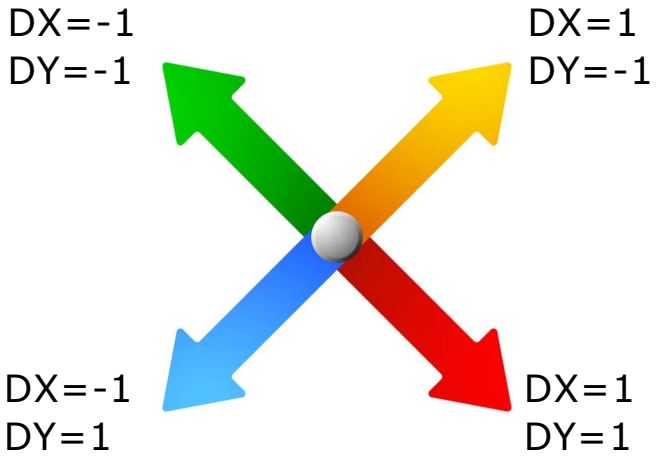
- Tenemos todos los ingredientes para poder hacer la función que compruebe si el jugador ha ganado:

- Si cualquiera de las funciones devuelve un 0, el jugador ha ganado.
  - checkHorizontal
  - checkVertical
  - checkDiagonals

Pero… esto es mejorable (opcional)

Generalización (OPCIONAL)

- Estas eran nuestras diagonales:
  - Diagonal 1: DX,DY = (-1,-1)(1, 1)
  - Diagonal 2: DX,DY = (-1, 1)(1, -1)



- Pero ¡podemos hacer horizontal y vertical de la misma forma!
  - Vertical: DX, DY = (0,-1)(0, 1)
  - Horizontal: DX, DY = (-1, 0)(1, 0)

¿Te animas a cambiar el código?