**Editor**: Fuentes en C o assembler

**Compilador**: Genera un objeto o si está configurado produce directamente el código de máquina. Traduce de un lenguaje a otro usualmente a ensamblador.

**Ensamblador**: Produce objetos en lenguaje de máquina a partir de assembler.

**#Directivas**:

**$ORG**: Permiten almacenar instrucciones en direcciones de memoria específicas.

**$DB**: Define Byte, asocia un valor a un símbolo

**$DW**: Define Word, asocia un valor a un símbolo

**$DL**: Define Long, asocia un valor a un símbolo

**$DS**: Define storage, permite reservar espacio.

**$EQU**: Asocia un valor cualquiera a un símbolo.

**$IF/ELSE/ENDIF**:

**$Include:**

**$DEFM**: Define macro

**$ENDM**: fin macro

**$Public**: Hace que un símbolo pueda ser usado por otra fuente.

**$Extern**: Permite usar el símbolo de una fuente de otra fuente.

**librarian**: Produce -lib. Toma los objetos, los organiza para que sea más fácil de utilizar. Crea una tabla de símbolos públicos que me permite buscar a los objetos.

**Encadenador**: Produce un ejecutable exe o .bin. Puede producir archivos de ayuda .map que dice como quedó organizado todo. Une todos los objetos tanto individuales o juntos en un .lib. El primer objeto es el startup siempre.Toma los segmentos y los ubica en sus respectivas zonas de memoria. Busca concordancias entre las tablas de símbolos públicos y externos para direccionarlos. Si quedan símbolos externos pendientes se revisa la lista de símbolos de los .lib e incluye sus respectivos segmentos en memoria de la misma forma.

**#Tabla de símbolos públicos**:

**#Tabla de símbolos externos**:

**make**: Controla la ejecución de todas las herramientas para producir el ejecutable.

**IDE**:

**Programador**:

**Simulador**: Software que se comporta como el chip a nivel funcional. No simula en tiempo real y por ende tampoco puede simular periféricos y su comportamiento.

**Emulador**: Hardware que se comporta como el chip. Son los mismos chips pero trabajan a velocidades más rápidas.

**#Pod**: La CPU está conectada a un bloque de mapeo que está conectada a una memoria de emulación o la memoria real. Este componente extra le permite configurar características básicas como si se desea acceder a RAM o ROM.

**#Analizador de bus**: Analizador de estados lógicos pero mas costoso y con más funcionalidades

**% Permite implementar watchpoint desde software lo cual no tiene ningún costo en tiempo pero es más costoso.**

**Depurador/ debugger**: Software que permite correr hasta cierto punto del programa o por línea y observar qué es lo que hace. La rutina del debugger gasta memoria.

**WatchPoint**: Busca monitorear una posición de memoria de forma que para cuando una posición de memoria es modificada.

**Breakpoint**:

**Desensamblador**: Toma código de máquina y lo transforma en assembler.

**Cargador**:

**Relocalizador**: Toma un programa y todas las direcciones de este y lo re-localiza en otra dirección diferente.

**Profiler**: Programa para analizar el tiempo en que se demora ejecutando el código, Calcula el mínimo, máximo y el promedio. Mide la frecuencia con la cual se ejecuta una rutina.

**Monitor**:

**compilador de C**:

**$preprocesador**: Sigue las directivas de C. Interpreta las directivas y entrega un código limpio de directivas para compilar.

**#Directivas**:

**$define**: Permite reemplazar ocurrencias de texto por valores. Permite definir constantes. Permiten generar macros.

**%legibilidad del código**:

**$if**: Permite generar verificaciones o generar diferentes modos de operación.

**$include**: Permite incluir fuentes, reemplaza la línea por un archivo entero.

**“”**: Está en la carpeta del proyecto

**<>**:Toca buscar el archivo a la lista de búsqueda.

**$else**:

**$error**: Le avisa al compilador que existe un error y que debe parar.

**$pragma**: Comodín para ejecutar directivas creadas por los creadores del compilador. Se usa usualmente para crear interrupciones u otros atributos.

**$elseif**:

**$compilador**:

C es un lenguaje multifuente, es decir, múltiples archivos

.h me permiten crear definiciones

**Objeto**: Archivo con instrucciones para reubicar instrucciones. Separan datos del programa.

**#Símbolos**:

**$Público**: Tienen acceso a todas las fuentes.

**$Externos:** Fuente externa

**#Segmentos**:

**#Datos**:

**#Programa**:

Las memorias tienen dos buses que permiten leer simultáneamente sobre las memorias.

\*\*\*\*

**Espacios de memoria/ Espacios de direccionamiento**:

**#**: Distingue rangos de direcciones y habilita los componentes necesarios

**$Memoria de datos:** Si es harvard

**$Memoria de programas:** Si es harvard

**$E/S:** Algunas arquitecturas son totalmente independientes a la memoria principal. Mantener buses independientes permiten mayor velocidad y permiten solventar variaciones altas de velocidades entre la cpu y los periféricos. Necesita instrucciones específicas.

**#Instrucciones específicas**:

**$In:**

**$Out:**

**Lenguajes de alto nivel**: Solo soportan espacios de memoria estándar. Ignoran los periféricos. Si se desea hacerlo compatible se usa macros que respeten las reglas de C y sustituirlas por equivalentes como input() y output() en ensamblador.

si se desea que E/S este en la memoria de datos es necesario que se encuentre siempre en la misma posición de memoria y es un problema porque el encadenador almacena objetos en posiciones aleatorias de memoria. La mejor solución es definir directivas con las posiciones de cada periférico en memoria.

**Memoria en C**:

**$Datos**:

#**Cualificadores**:

**$Automaticas**: Se encuentran en la pila.

**$Register**: Se almacena en un registro, si no alcanza el número de registro, se almacena en la pila.

**$Static**: Se le asigna una posición fija de memoria en la memoria ram y existe todo el tiempo sin importar los corchetes o estructuras del código. Si se asigna una variable fuera de los corchetes se convierte en una variable estática y pública, informalmente están asociadas a las variables globales. Definiendo explícitamente una rutina o una variable evita que esta última sea pública.

**$Dinamica**: Las variables se asignan en el bloque HEAP. Las variables asignadas fragmentan el bloque heap. Esto incrementa los tiempos de asignación de variables mientras busca espacio idóneo. Evita usar demasiada memoria.

**#Malloc**:

**#Free**:

**$extern**: Trae una variable de otra fuente.

**$Volatile**: Evita que el compilador asuma un valor constante a una variable concreta.

**$const**: Define constantes (solo para el compilador), el usuario puede modificarlo a través de un apuntador castiandolo como un apuntador tradicional.

**$far**: No es estándar, diferenciar apuntadores largos.

**$near**: No es estándar, diferenciar apuntadores cortos.

.

**Consideraciones de eficiencia**:

**#CPU**: El recurso más importante de CPU es el tiempo. NO ponerla hacer por hacer. Se debe poner énfasis en el trabajo útil.

**#Periféricos**: El recurso más importante es el tiempo de respuesta, el código debe ser lo suficientemente rápido para atender a los periféricos con los tiempos de respuesta que estos plantean.