Ciclo de desarrollo y Lenguaje ensamblador

- Lenguajes
  - Cód. Máquina
  - Lenguajes alto nivel
  - Lenguaje ensamblador
- Ciclo de desarrollo
- Lenguaje ensamblador

Lenguajes
Código de máquina

- Nivel de abstracción bajo
  - ceros y unos.
- Es lo único que entiende el procesador.
- Etapa final de todo desarrollo:
  - código de máquina en memoria.
- Imposible entender, menos aún corregir o modificar por un humano.

```
Cód. máquina
0011 1010
0000 1100
0000 0000
0100 0111
0011 1010
0000 1101
0000 0000
1000 0000
0011 0010
0000 1110
0000 0000
0111 0110
```

Lenguajes
Lenguajes alto nivel

- Pascal, C, Java

```
Suma := Oper1 + Oper2
```

- Mayor nivel de abstracción
- No lo comprende el procesador
  - Necesita Traducción (Compilación)
  - Traductor = Compilador
- Independiente del computador elegido (Portable).

Lenguajes
Lenguaje ensamblador

- Nivel de abstracción intermedio
- Asociado al procesador usado
- Traducción más simple
  - Una línea corresponde a una instrucción
  - Traducción = Ensamblado
  - Compilador = Ensamblador (igual que el lenguaje)

Código máquina	Ensamblador	Alto nivel
3A 01 B2	LD A, (OPER1)	SUMA := OPER1 + OPER2
47	LD B, A	
3A 02 B2	LD A, (OPER2)	
80	ADD A, B	
32 00 B2	LD (SUMA), A	

Lenguajes
Lenguaje ensamblador

- Elementos
  - Mnemonics
  - Símbolos asociados a:
    - Direcciones (etiquetas)
    - Constantes

Código máquina	Ensamblador	Alto nivel
3A 01 B2	LD A, (OPER1)	SUMA := OPER1 + OPER2
47	LD B, A	
3A 02 B2	LD A, (OPER2)	
80	ADD A, B	
32 00 B2	LD (SUMA), A	

Lenguajes
Cuándo usar ensamblador?

- Alto nivel siempre que sea posible
- Motivos para usar assembler:
  - Acceso a todos los recursos del procesador
  - Eficiencia
    - en tiempo de ejecución
    - en tamaño de código
  - No disponibilidad de compilador
- A menudo mixto
  - Secciones críticas en assembler
  - Resto en alto nivel

Ciclo de desarrollo y Lenguaje ensamblador

- Lenguajes
- Ciclo de desarrollo
  - Diseño
  - Codificación
  - Traducción
  - Prueba y depuración
  - Documentación y mantenimiento
- Lenguaje ensamblador
  -

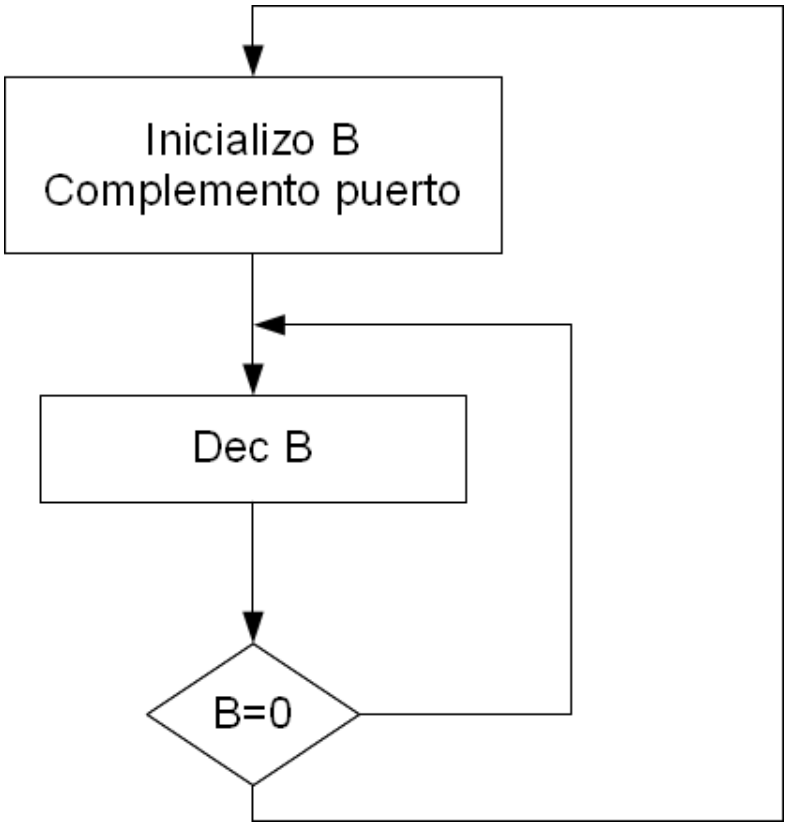
Ciclo de desarrollo
Diseño

- Nunca empezar escribiendo código
  - *“Spaghetti code”*
  - Empezar con seudocódigo o diagramas de flujo
- Seudocódigo:
  - estructuras de control de lenguajes de alto nivel pero con libertades para incluir recursos hardware

```
mientras (true){
    complemento bits de salida
    b = semiperíodo
    mientras ( b !=0 ) {
        decremento b
    }
}
```

Ciclo de desarrollo
Diseño

- Diagramas de flujo
  - Menor nivel de abstracción
  - Más fácil pasarlo a assembler
  - Más confuso para programas complejos
- Usen el que prefieran
  - Pero usen!!
  - Caso contrario
    - *“Spaghetti code”*
    - Difícil de entender por otros
    - Difícil de entender por el propio desarrollador más adelante

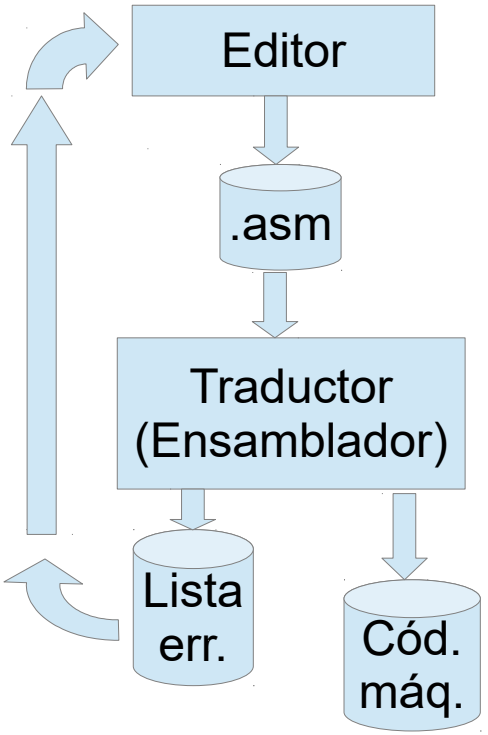


Ciclo de desarrollo
Diseño

- No fui claro?
- Usen lo que prefieran, seudocódigo o diagrama de flujo
  - pero USEN !!

Ciclo de desarrollo

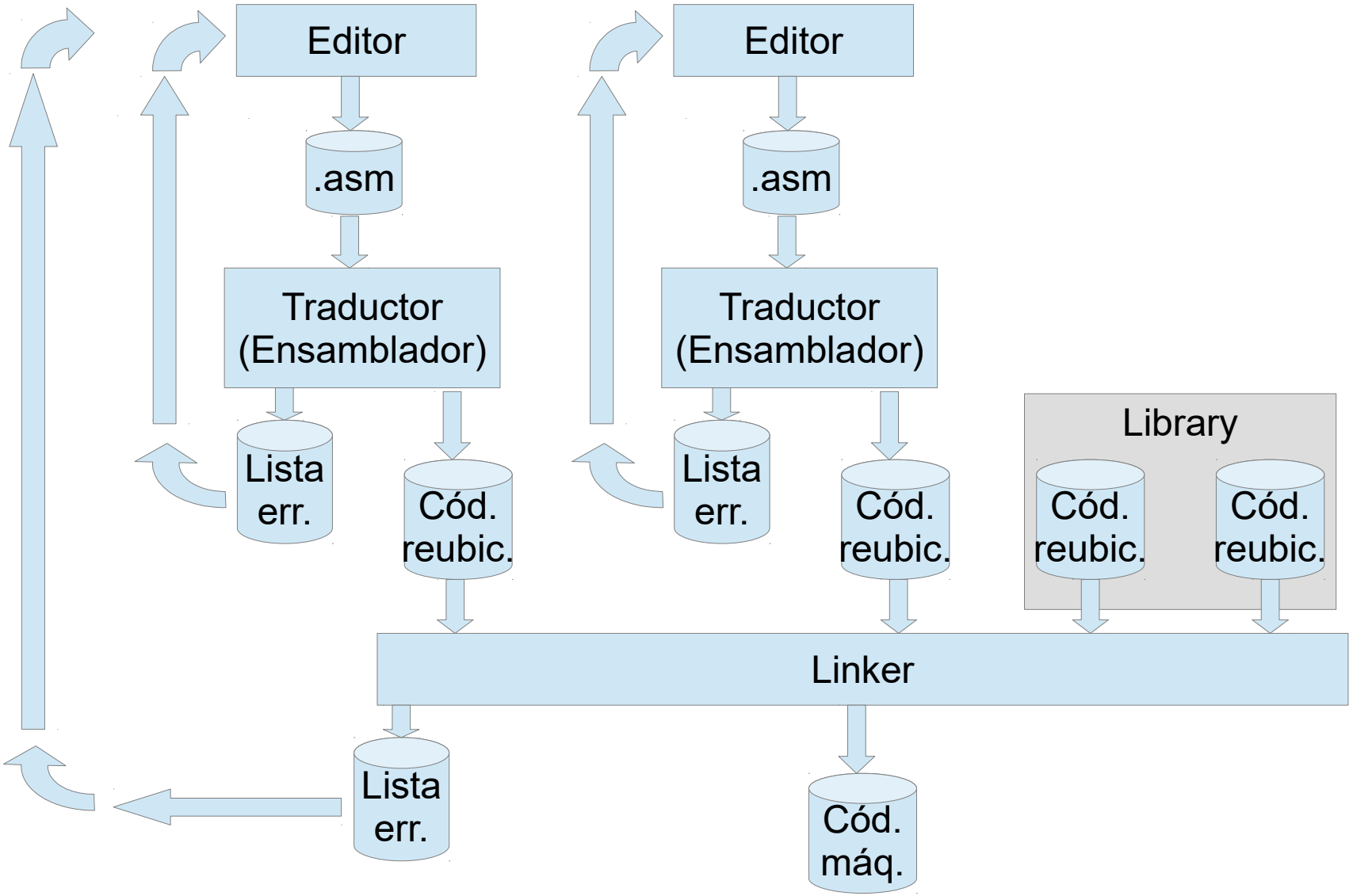
- Codificación
  - Después de terminado diseño
  - Editor de texto: archivo fuente
  - En este caso lenguaje assembler
- Traducción
  - Ensamblado
  - Lista de errores
  - Si es posible: código máquina



Ciclo de desarrollo
Traducción - Linker

- Programa en archivos separados
- Assembler
  - Traduce cada módulo
  - Direcciones relativas al comienzo del módulo
  - Símbolos externos (definidos en otro módulo)
- Linker
  - Concatena (*encadena*) los módulos
  - Resuelve pendientes
    - Direcciones de comienzo ahora conocidas
    - Todos los símbolos tienen que estar definidos en algún módulo
- Bibliotecas (*library*) ya ensambladas
  - De terceros o propias

Ciclo de desarrollo
Traducción - Linker



Ciclo de desarrollo y Lenguaje ensamblador

- Lenguajes
- Ciclo de desarrollo
  - Diseño
  - Codificación
  - Traducción
  - Prueba y depuración
  - Documentación y mantenimiento
- Lenguaje ensamblador
  -

Ciclo de desarrollo
Prueba y depuración

- Objetivo: verificar que el programa hace lo especificado
- Dos enfoques:
  - Teórico:
    - Modelos y prueba matemática (verificación formal)
  - Enfoque tradicional:
    - Ejecución controlada del programa
    - Comparación de resultados con resultados esperados
- Cargador (Loader)
  - Carga el código máquina en memoria
- Depende del ambiente de desarrollo
  - Sistema de desarrollo: donde escribo y pruebo los programas
  - Sistema destino: donde va a correr el programa final

Prueba y depuración Herramientas

- Simulador
- Depurador o *Debugger*
- Sistema destino
  - Kit de desarrollo
  - Placa prototipo
- Analizador Lógico
- Caso laboratorio
  - Sistema de desarrollo: PC
  - Sistema destino
    - Simulador
    - Kit de desarrollo en placa DE0

Ciclo de desarrollo y Lenguaje ensamblador

- Lenguajes
- Ciclo de desarrollo
  - Diseño
  - Codificación
  - Traducción
  - Prueba y depuración
  - Documentación y mantenimiento
- Lenguaje ensamblador
  -

Ciclo de desarrollo
Documentación y Mantenimiento

- Documentación
  - De diseño (seudocódigo, diagramas de flujo)
  - De código: comentarios
- Necesario
  - Para ser entendido por terceros
  - Ídem por el mismo desarrollador tiempo más tarde
- Mantenimiento durante vida del código
  - Corrección de bugs
  - Nuevas funcionalidades
  - Reutilización del código

Documentación y Mantenimiento

- Documentación
  - De diseño (seudocódigo, diagramas de flujo)
  - En el código: comentarios
- Importante para
  - Que sea entendible por terceros
  - Ídem por el mismo desarrollador un tiempo después
- Durante la vida del programa
  - Corrección de bugs
  - Nuevas funcionalidades
  - Re-uso del código

Lenguaje ensamblador

- Mnemonics
- Símbolos para referir a:
  - Direcciones (etiquetas)
  - Constantes
- Expresiones
  - Pueden contener:
    - Constantes
    - Identificadores
    - Operaciones: NOT, AND, OR, XOR, +, -, *, /, MOD, …
  - Evaluadas en tiempo de ensamblado por el ensamblador
    - NO en tiempo de ejecución por el procesador
  - Ejemplo: Si el identificador “DIR” tiene el valor 0x8010.
    - LD H, DIR / 256 equivale a LD H,0x80
    - LD L, DIR MOD 256 equivale a LD L, 0x10

Proceso de ensamblado

- Contador de posiciones
  - Variable del algoritmo de traducción ($)
  - Dirección (absoluta o relativa al comienzo) donde debe cargarse la traducción de esa línea de código.
  - NO ES el Contador de Programa!!
    - NO existe en tiempo de ejecución.
- Etiquetas
  - Símbolo. Se asocia al valor de $ en esa línea
- Tabla de símbolos
  - Pares símbolo-valor
  - Definidos mediante etiquetas o directivas

Proceso de ensamblado Algoritmo en dos pasadas

- Primera pasada
  - Se calcula contador de posiciones en cada línea
  - Se agregan a tabla de símbolos etiquetas y otros símbolos
- Segunda pasada
  - Opcode y parámetros según cartilla
  - Símbolos se sustituyen por su valor según tabla
- Si direcciones son relativas queda pendiente para el *linker* sumar dirección de comienzo
- Idem con símbolos externos

Proceso de ensamblado
Algoritmo en dos pasadas

```
   1
   2
   3
   4
   5
   6
   7                        onda:    ld B, semiper
   8                                 CPL
   9                                 out (salida), a
  10                        lazo:    dec b
  11                                 jp nz, lazo
  12                                 jp onda
  13
  14

DEFINED SYMBOLS
```

Proceso de ensamblado
Algoritmo en dos pasadas

```
   1
   2                        ;Definicion de simbolos
   3                        salida    equ    0xFF
   4                        semiper   equ    10
   5
   6                            .text
   7 0000                   onda:    ld B, semiper
   8 0002                            CPL
   9 0003                            out (salida), a
  10 0005                   lazo:    dec b
  11 0006                            jp nz, lazo
  12 0009                            jp onda
  13
  14                                 .end

DEFINED SYMBOLS
                onda.s:7        .text:00000000 onda
                onda.s:10       .text:00000005 lazo
```

Proceso de ensamblado
Algoritmo en dos pasadas

```
   1
   2                        ;Definicion de simbolos
   3                        salida    equ    0xFF
   4                        semiper   equ    10
   5
   6                            .text
   7 0000 060A              onda:    ld B, semiper
   8 0002 2F                         CPL
   9 0003 D3FF                       out (salida), a
  10 0005 05                lazo:    dec b
  11 0006 C205 00                    jp nz, lazo
  12 0009 C300 00                    jp onda
  13
  14                                 .end

DEFINED SYMBOLS
                 onda.s:7       .text:00000000 onda
                 onda.s:10      .text:00000005 lazo
```

Lenguaje ensamblador

- Formato de una sentencia:

  [etiqueta:] [instrucción o directiva] [comentario]

- Los tres campos son opcionales
- Separadores usuales
  - ":" para fin de etiqueta
  - ";" o "//" para inicio de comentario
  - Algunos permiten /* comentarios en bloque que pueden ser largos y extenderse a más de una línea */
- Constantes numéricas
  - Decimales: 15 (en general opción por defecto)
  - Binario: 00001111B (ojo!! solo B mayúscula en Gnu assembler del lab.)
  - Hexadecimal: 0x0F (algunos aceptan 0Fh, pero no Fh porque lo confunde con un símbolo)

Lenguaje ensamblador Directivas

- ORG (Origen)
  - Indica a partir de qué dirección debe cargarse el código que sigue a continuación.
  - Asigna valor a contador de posiciones
  - Sintaxis:

```
ORG <expresion>
.org <expresion>
```

  - Dos variantes
    - Absoluto
    - Relativo

Lenguaje ensamblador Directivas

- ORG (continuacion)
  - Gnu assembler usado en el curso
    - Solamente relativo
    - Algoritmo en una sola pasada => Directivas ORG deben estar en orden creciente de direcciones en el archivo
    - Secciones
      - .text usualmente código y constantes
      - .data usualmente para variables
      - Son reubicables, se define al invocar al *linker* donde comienza cada una.

Lenguaje ensamblador Directivas

- EQU o EQUATE
  - Correspondencia símbolo – valor
  - Sintaxis usuales
    - <simbolo> EQU <valor>
    - .equ <simbolo>,<valor>
  - Al procesar esta línea se agrega el par símbolo-valor a la tabla de símbolos
  - Es la forma de definir semiper en el ejemplo

Lenguaje ensamblador
Reserva de memoria

- Define byte

  <etiqueta>: DB [valor]

  <etiqueta>: DEFB [valor]

  <etiqueta>: .byte [valor]

  - Incrementa en 1 contador de posiciones para dejar lugar para un byte
  - Si se incluye [valor] se pone ese valor como resultado de traducción.
- Gnu assembler:
  - Bug. Obliga a poner un valor aunque luego no se utilice

Lenguaje ensamblador
Reserva de memoria

- Define word

  <etiqueta>: DW [valor]

  <etiqueta>: DEFW [valor]

  <etiqueta>: .hword [valor]

  - Ídem reservando dos bytes

- Define storage

  <etiqueta>: DS tamaño [valor]

  - Parametro adicional para especificar tamaño

Lenguaje ensamblador Directivas

- Terminar traducción (END)

  .end

  - El texto que viene después es ignorado por el traductor.
  - Gnu assembler:
    - Se debe terminar la línea con retorno de carro, de lo contrario no la procesa
- Incluir otro archivo (.include)
  - .include <*nombre_archivo*>
  - *I*ntercala contenido de *nombre_archivo*
  - Cómodo para tener una sola versión de código usado en diferentes archivos (p. ej. laboratorio)