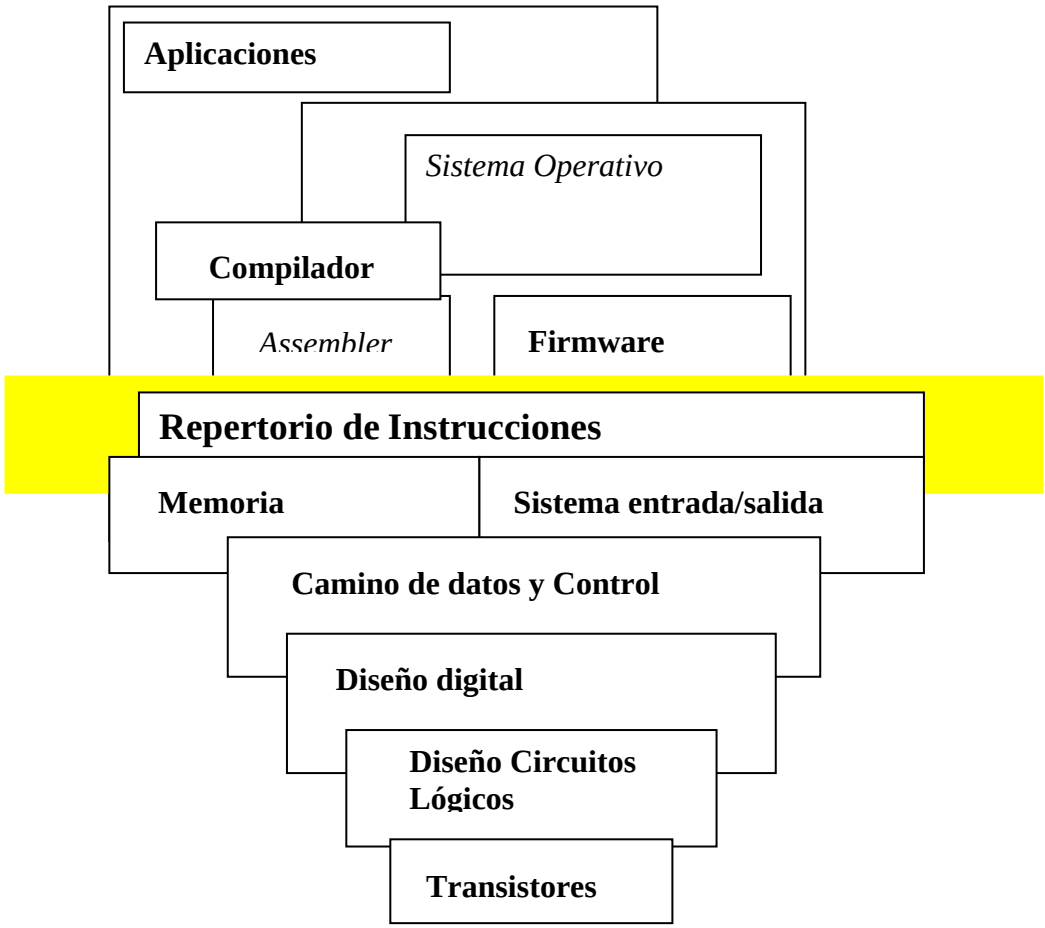
1. Introducción.

La descripción de un computador digital puede efectuarse mediante la consideración de diferentes niveles de abstracción.

La siguiente figura ilustra diversas componentes abstractas:



Todas las partes están sometidos a rápidos cambios debidos a la tecnología, los ambientes de programación, las aplicaciones, los sistemas operativos.

1.1. Repertorio de Instrucciones.

Se destaca el repertorio de instrucciones, ubicado en un nivel medio, siendo la interfaz entre la zona superior que podríamos denominar la parte software, y la parte inferior que suele denominarse hardware.

El repertorio de instrucciones establece la interfaz software-hardware.
Esta interfaz es fundamental en el diseño de un computador.

El repertorio de instrucciones es el enfoque o visión que tienen de un computador los Diseñadores de Compiladores y Sistemas Operativos.
La arquitectura del repertorio establece los atributos del sistema computacional vistos por el programador.

La elección de la arquitectura del repertorio se refleja en:

- La organización del almacenamiento:
  - Tipos de Datos
  - Estructuras de Datos
  - Codificación y Representaciones.
- Repertorio de Instrucciones
- Formatos de Instrucciones
- Modos de direccionamiento y formas de accesar a los datos y a las instrucciones
- Tratamiento de Excepciones.

1.2. Tipos de repertorios.

Se clasifican de acuerdo a cómo se especifican los operandos de entrada y de salida.

Los ejemplos siguientes se ilustran con la operación suma. Obviamente en los diferentes tipos de repertorios existen más instrucciones que las aritméticas.

1.2.1. Máquinas de Stack.

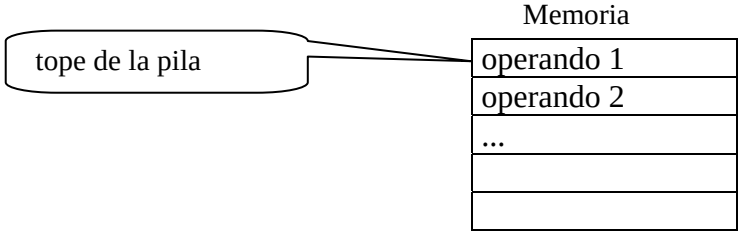
Se denomina de 0 direcciones. Ya que sólo se especifica la operación a realizar con los datos, los cuales se encuentran ubicados en posiciones fijas de la memoria.

add

La cual se interpreta según:

$$M[tope-1] \Leftarrow M[tope-1] + M[tope]; tope--;$$

Donde M[dir] corresponde al valor almacenado en la dirección dir de la memoria.
Se requiere especificar sólo la operación a realizar. Se omiten los campos de los operandos de entrada y salida. Se asume que los operandos están en posiciones fijas de la memoria, y por lo tanto no se necesitan bits para especificarlos.



Se asume que las direcciones aumentan hacia arriba en el diagrama.

1.2.2. Máquinas con acumulador.

Se denominan también de 1 dirección. Se especifica una dirección de un operando, el otro se encuentra en un registro fijo denominado acumulador. El resultado se deposita en el acumulador.
Entonces, la instrucción:

add A

se interpreta según:

acc ⇐ acc + M[A]

Nótese que A, es el símbolo de una dirección.

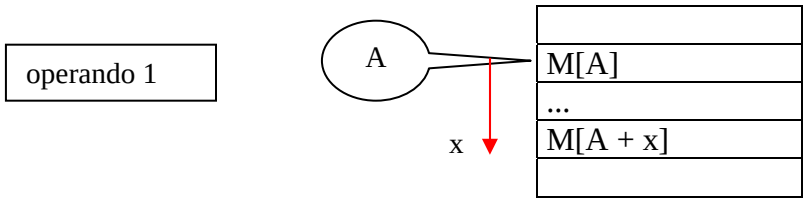
Una variante es especificar una dirección y un valor constante.
Una dirección con offset:

add x(A)

Cuya acción es:

acc ⇐ acc + M[A + x]

Por defecto el resultado y uno de los operandos de entrada se encuentra en un registro especial, denominado acumulador.



Por lo tanto sólo se requiere especificar una dirección, A en este caso.

1.2.3. Máquinas de Registro-Memoria.

Se denominan de 2 direcciones. Los operandos pueden ser direcciones de memoria o registros. Se denomina dirección efectiva aquélla donde reside el valor que será procesado.
El operando del resultado es uno de los de entrada, ya que se tienen dos direcciones:

add opA, opB

Que efectúa:

opA ⇐ opA + opB

Si se especifica el lugar en donde depositar el resultado, tenemos tres direcciones:

add opA, opB, opC

Que realiza:

opA ⇐ opB + opC

En algunas implementaciones, se interpreta el orden de los operandos así:

opC ⇐ opA + opB.

Sea por ejemplo el operando A un registro (R1) y el operando B una dirección de la memoria (dir2), entonces:

add R1, dir2

Se interpreta: R1 = R1 + M[dir2]

Si los dos operandos son direcciones de celdas de memoria, la instrucción:

add dir1, dir2

se interpreta: M[dir1] = M[dir1] + M[dir2]

1.2.4. Máquinas de Carga –Almacenamiento. (Load-Store)

Se especifican 3 direcciones.

add	Ra, Rb, Rc	Ra ⇐ Rb + Rc
load	Ra, Rb	Ra ⇐ M[Rb]
store	Ra, Rb	M[Rb] ⇐ Ra

Las operaciones se efectúan solamente entre registros. La única forma de accesar la memoria es vía registros, para ello se dispone de las instrucciones load y store.

1.3. Comparación de las arquitecturas.

Las diferentes arquitecturas de repertorios suelen compararse, observando:

El número de bytes por instrucción, el número de instrucciones que se requieren para efectuar una acción determinada, el número de ciclos de reloj por instrucción.

Por ejemplo, para realizar C = A + B; donde A, B y C son variables en memoria, se requieren las siguientes secuencias de instrucciones, dependiendo del tipo de repertorio:

a) Máquina de stack:

```
Push    A       ; tope++, M[tope] = M[A]
Push    B       ; tope++, M[tope] = M[B]
Add             ; M[tope-1] += M[tope], tope--
Pop     C       ; M[C] = M[tope], tope—
```

Donde push y pop son instrucciones del repertorio de una máquina de stack, para movilizar datos desde la memoria hacia el tope del stack y viceversa.

La variable tope se emplea aquí como un puntero (o sea almacena una dirección) que apunta a la última dirección ocupada en el stack.

b) Máquina con acumulador:

```
Load    A          ; acc = M[A]
Add     B          ; acc = acc + M[B]
Store   C          ; M[C] = acc
```

Donde load y store son instrucciones para mover datos entre la memoria y el acumulador y viceversa.

c) Máquina con operandos registro-memoria

```
Load    R1, A      ; R1 = M[A]
Add     R1, B      ; R1 = R1 + M[B]
Store   R1, C      ; M[C] = R1
```

d) Máquina con registros carga-almacenamiento

```
Load    R1, A      ; R1 = M[A]
Load    R2, B      ; R2 = M[B]
Add     R3, R1, R2 ; R3 = R1 + R2
Store   R3, C      ; M[C] = R3
```

1.4. Instrucciones típicas de un repertorio.

Se enumeran operaciones típicas que se encuentran en casi todos los repertorios de instrucciones desde los años 60. No todas ellas están presentes en un repertorio determinado. En caso de no estarlo pueden implementarse con las primitivas existentes. Los ensambladores con capacidad de crear macros, permiten que el propio programador cree secuencias especiales, que le permitan programar con un conjunto ampliado de instrucciones.

Se agrupan por tipos

- **Movimientos de Datos:**
  Cargar registro desde la memoria.
  Almacenar registro en la memoria.
  Movimiento entre celdas de memoria.
  Movimiento entre registros.
  Desde registro hacia dispositivo de salida (output).
  Desde dispositivo de entrada hacia registro (input).
  Push, pop. Operaciones en stack (pila).

- **Aritméticas:**
  Suma, resta, multiplicación, división. En binario, con y sin signo.
  Suma, resta, multiplicación, división. En BCD, con y sin signo.
  Suma, resta, multiplicación, división. En punto flotante (reales binarios), con y sin signo.
  Suma, resta, multiplicación, división de precisión extendida. En binario, con y sin signo.

- **Corrimientos:**
  Aritméticos hacia la izquierda y derecha.
  Lógicos hacia la izquierda y derecha.
  Rotaciones hacia la izquierda y derecha.

- **Control de Secuencia de ejecución:**
  Comparaciones.
  Saltos condicionales e incondicionales.
  Bifurcaciones condicionales.

- **Enlace a Subrutinas:**
  Llamado y retorno de subrutinas.

- **Interrupciones:**
  Programación del sistema de prioridades, habilitación y deshabilitación de las interrupciones, retorno de interrupciones enmascaradas y no enmascaradas.

- **Sincronización:**
  Test and set. Operación para manejar procesos.

- **Control del procesador:**
  Halt, manejo de memoria virtual y caché, llamados al sistema, entrada y salida de modo supervisor.

- **String:**
  Traducir, buscar, etc.

- **Gráficas:**
  Operaciones para el manejo gráfico.

1.5. Estadísticas de uso de instrucciones.

Se muestra a continuación una estadística de la frecuencia de uso de instrucciones de un procesador Intel (8086), confeccionada contando las instrucciones que aparecen en un programa tipo, empleando sólo operaciones con enteros.

Existen programas especiales para efectuar estas estadísticas.

Nótese que son dominantes las frecuencias de las instrucciones simples. El conjunto mostrado, ilustra que sólo el 5 % de las instrucciones no pertenecen a las mostradas.

load	22 %
bifurcación condicional	20 %
compare	16 %
store	12 %
add	8 %
and	6 %
sub	5 %
movimiento entre registros	4 %
llamados a subrutinas	1 %
retornos de subrutinas	1 %

La tabla anterior, justifica empíricamente los repertorios reducidos de instrucciones (RISC).

1.6. Modos de direccionamiento.

Se describen resumidamente, diferentes modos de direccionamiento. No todos están presentes en los diferentes repertorios

Modo	**Ejemplo**	**Significado**
Registro	Add R4, R3	R4 ⇐ R4 + R3

Los operandos se encuentran en registros.

Inmediato	Add R4, #3	R4 ⇐ R4 + 3

Un campo contiene una constante, generalmente un número con signo. Es decir, un número que está contenido en la instrucción.

Desplazamiento	Add R4, 100(R1)	R4 ⇐ R4 + M[100 + R1]

Permite direccionar un elemento de una estructura (struct en C, record en Pascal) cuyo inicio esté apuntado por el registro R1. El desplazamiento (offset) es respecto a un registro base.
Nótese que la dirección efectiva es 100 + R1.

En el caso de instrucciones de salto, suele efectuarse saltos relativos al registro PC, empleando el valor inmediato como desplazamiento.

Registro indirecto Add R4, (R1) R4 ⇐ R4 + M[R1]
Permite direccionar arreglos si en R1 se mantiene la dirección de la componente. Si se desea accesar otra componente debe cambiarse el contenido del registro R1.

Se denomina indirecto ya que el dato no se encuentra en R1 sino en la dirección apuntada por R1. El registro R1 es un puntero, ya que es una variable que contiene una dirección.

Indice / Base Add R3, (R1 + R2) R3 ⇐ R3 + M[R1 + R2]
Es el direccionamiento que permite tratar en forma flexible arreglos y estructuras. En un registro base se coloca la dirección inicial del elemento. En el registro índice se almacena el desplazamiento de la componente, relativo a la base.

Directo o absoluto Add R1, (1001) R1⇐ R1 + M[1001]
Un campo contiene una dirección de memoria.

Memoria indirecto Add R1, @(R3) R1 ⇐ R1+ M[M[R3]]
La dirección efectiva se encuentra en M[R3].

Auto-incremento Add R1, (R2)+ R1⇐ R1 + M[R2]; R2 ⇐ R2+d
Se efectúan dos acciones, la operación de sumar, y el incremento del registro R2 en una cierta cantidad fija d.

Auto-decremento Add R1, –(R2) R2 ⇐ R2 – d; R1 ⇐ R1+ M[R2]

Escalado Add R1, 100(R2)[R3] R1⇐R1+ M[100+R2+R3*d]

Uso de Registros.
Todas las máquinas modernas usan **múltiples registros**. Esto debido a que los registros son más rápidos que la memoria principal y que la memoria caché; además permiten una mejor compilación de expresiones (es decir la traducción a una secuencia de instrucciones de máquina) que una evaluación en una pila.

Su principal ventaja es que pueden mantener las variables de uso frecuente, rápidamente accesibles, reduciendo los accesos a memoria.

En el lenguaje C existe la posibilidad de indicar al compilador cuáles variables deben ser almacenadas en registros, agregando la palabra *register* a la lista de variables.

Otra ventaja importante en los mecanismos de direccionamiento es que se requieren menos bits para especificar registros que para especificar direcciones de memoria.

Debido a técnicas de segmentación se suelen emplear repertorios tipo carga-almacenamiento, como se verá más adelante.

1.7. Estadística de uso de modos de direccionamiento.

A través de mediciones, en máquinas que tienen todos los modos de direccionamiento, y con diversos programas, se llegó a que:

un 42% de los modos usados son con offset,
un 33% inmediatos,
un 13% son con registro indirecto,
un 12% de otros modos.

Es notable que el 88 % de los modos de direccionamiento empleados correspondan a modos con desplazamiento, inmediatos y con registro indirecto. Esto implica que en cualquier diseño actual dichos modos deben estar presentes.

Nótese que con desplazamiento cero se logra el direccionamiento de registro indirecto. Siempre a través de mediciones se concluyó que los tamaños de los valores inmediatos usados en un 50% al 60% de los casos puede lograrse con 8 bits (números desde 0 hasta 256, o bien entre –128 y +127); y que con 16 bits para el campo inmediato se pueden efectuar hasta un 80% de los casos de direccionamiento inmediato.

También por mediciones, se determinó que los desplazamientos, empleando 16 bits permiten cumplir el 99% de los casos. Sólo un 1 % de las referencias requieren más de 16 bits.

1.8. Organización de máquina.

Desde un punto de vista de los diseñadores del hardware el estudio de un computador describe la forma en que están organizados los flujos de datos, la estructura del control y la ejecución de microinstrucciones. También incluye el diseño lógico (usando componentes digitales) y la implementación física.

Desde un punto de vista del diseñador lógico, interesan:

- Las capacidades y comportamiento característico de las principales unidades funcionales. (por ejemplo: Registros, ALU, Shifters, Unidad de control ...)
- Las formas en que esas componentes están interconectadas.
- Los flujos de información entre las componentes.
- Medios lógicos por los cuales se controlan los flujos de datos.
- Cómo las unidades funcionales (o recursos) realizan el repertorio de instrucciones.
- Lenguaje de transferencia entre **registros.**

1.9. Partes de un computador.

° Todos los computadores consisten de 5 partes:
- Procesador: (1) camino de datos y (2) control
- (3) Memoria
- (4) Dispositivos de entrada y (5) Dispositivos de salida

° No todas las memorias tienen la misma construcción y estructura
- Caché: es una memoria rápida en tiempo de acceso, pequeña en capacidad, tiene costo elevado y está ubicada cerca del procesador
- Memoria principal: es más lenta y de mayor volumen en capacidad de almacenamiento(es más barata que la anterior).

° Los dispositivos de Entrada-Salida tienen diversas organizaciones, ya que existen en un amplio rango de velocidades (red, pantalla, discos, mouse, teclado, etc.) y en un amplio rango de requerimientos (velocidades, costos, estandarizaciones, etc.)

1.10. Niveles de representación.

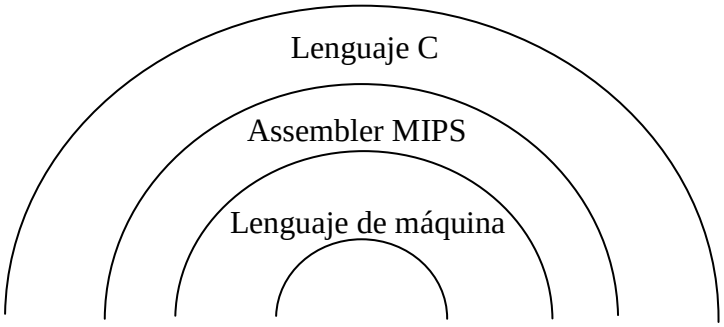
Precisaremos ahora el enfoque que emplearemos para comprender el funcionamiento y los principios de diseño de un computador. Nos concentraremos primero en un nivel inmediatamente superior al del repertorio de instrucciones, que corresponde a software, para ir gradualmente descendiendo en el nivel de abstracción.

Cada nivel tiene un lenguaje asociado. A medida que se desciende aparecen nuevos niveles de detalle, que explican en un nivel más bajo lo que se efectúa en el nivel alto. Estos dos puntos de vista se refuerzan entre sí y permiten entender lo abstracto a través de situaciones cada vez más concretas.

En cualquier lenguaje existe un aspecto **léxico** (vocabulario, las palabras que se pueden escribir); un aspecto **sintáctico** (reglas gramaticales, para escribir correctamente); y finalmente un aspecto **semántico** (que asigna un significado a las construcciones).

Un lenguaje de programación permite escribir programas que instruyen al computador sobre el conjunto organizado de *acciones* (algoritmo) que deben efectuarse sobre los *datos*.

Las acciones y los datos deben describirse con rigurosidad para que puedan ser correctamente interpretados por un autómata.



Decidimos comenzar en el nivel del lenguaje de programación C, para ir descendiendo. Los niveles más altos se alejan bastante del objetivo de lograr una explicación del funcionamiento de un procesador.

Estudiaremos primero la forma de describir datos y constantes en C y en assembler. En la medida que aparezcan conceptos, se los irá describiendo. Luego estudiaremos las acciones y el diseño de funciones.

1.10.1. Programación en lenguaje C.

Se emplea el lenguaje C, ya que éste fue diseñado considerando que sus elementos pudieran ser eficientemente traducidos a assembler.

La realización del recorrido que efectuaremos también permitirá aprender con mayor profundidad la programación en C, ya que veremos cómo se desglosa cada instrucción de C mediante una o varias instrucciones de un determinado assembler.

1.10.2. Programación assembler.

Comprender el funcionamiento de un computador puede lograrse conociendo las operaciones éste que puede realizar.

La descripción de las acciones que un procesador realiza y los datos que puede manipular pueden explicarse mediante instrucciones assembler.

El lenguaje assembler es la representación simbólica de la codificación binaria de las instrucciones o lenguaje de máquina.

El procesador decodifica los campos binarios y ejecuta la secuencia de transferencias que realizan las acciones que interpretan la instrucción.

En assembler se emplean símbolos para los códigos binarios: del tipo de instrucción, de los registros, de los valores constantes. También emplea símbolos para especificar direcciones de memoria donde pueden estar almacenadas instrucciones o datos.

Una instrucción requiere especificar las siguientes informaciones: Dónde se encuentran los operandos de entrada, cuál es la operación a realizar, dónde se depositará el resultado y cuál es la próxima instrucción a realizar. Cada uno de estos datos demandará una cantidad de información, que suele medirse en bits.

Por ejemplo: si los datos están en un arreglo de 32 registros, se necesitarán 5 bits para especificar el número de un registro; si la memoria tiene 1024 palabras (1 K[palabras]), la especificación de una dirección de una palabra requiere de 10 bits; si la memoria tiene 1 M[palabras], una dirección requiere ahora 20 bits. Si existen 16 tipos de operaciones, se requieren 4 bits para especificar la operación.

La forma específica de encontrar los operandos se fue flexibilizando y complicando a través del tiempo, ya que empleando algunos bits adicionales puede especificarse, por ejemplo, que la dirección de la memoria está en un registro, o que el operando se encuentra sumando el contenido de dos registros. Las diversas formas de interpretar la información para obtener los operandos se tratará, más adelante, bajo el título “modos de direccionamiento”.

Cada una de estas informaciones puede verse como un campo de bits:

operación	operando	operando	resultado	modo

Se ha ilustrado los campos de una instrucción con dos operandos de entrada y un resultado.

Si la instrucción es de salto incondicional, los campos de entrada y salida no estarán presentes, pero se tendrá un campo para especificar la dirección de salto.

Si la instrucción es una bifurcación (por ejemplo: salte a tal dirección si el operando uno y el operando dos son iguales) no se requiere el campo de resultado, pero éste se reemplaza por la dirección de salto.

Algunas instrucciones permiten que uno de los campos sea una constante. Se dice que estos campos son inmediatos (el valor viene inmediatamente en la instrucción). Si el dato está en un registro, se requiere la operación adicional de leer el registro; en estos casos se dice que el dato es indirecto (no viene directamente en el código de la instrucción).

Si la operación es monádica, no estará presente uno de los operandos de entrada (por ejemplo la negación bit a bit de un operando).

En los diferentes repertorios el orden de los campos puede ser distinto; en algunos otros, el campo de operación puede estar subdividido en dos.

Finalmente podemos conceptualizar una instrucción de máquina como una secuencia de unos y ceros. Si el número de bits necesarios es menor o igual al de una palabra de memoria, pueden *ensamblarse* los distintos campos en una celda de la memoria, como se ilustra a continuación.

instrucción

Si el número de bits es mayor, pueden emplearse dos o más palabras para especificar una instrucción. Esto implica que tendrá que accesarse en más de una oportunidad a la memoria para comenzar a procesar la instrucción.

La forma usual de especificar los operandos en assembler es emplear notación simbólica.

El código de operación suele describirse con una palabra (en inglés) que describe la operación; a este campo suele denominarse mnemónico. Suele ser una abreviatura o las primeras letras de un concepto; por ejemplo: jal que recuerda **j**ump **a**nd **l**ink.
Luego, si están presentes, le siguen los operandos de entrada y salida.

Suele describirse cada instrucción en una línea, y los campos suelen separarse con espacios (y más generalmente por tabs, lo que se logra digitando la tecla a la izquierda de la letra Q).

También al inicio de la línea puede existir un rótulo (label en inglés) que identifica, en forma simbólica, la dirección de la memoria en que comienza la instrucción. La separación entre el rótulo y el mnemónico de la operación suele ser dos puntos.

En este *lenguaje simbólico de máquina* puede definirse líneas completas como comentarios, y también puede colocarse un comentario de fin de línea después de los campos de la instrucción.

Entonces, puede generalizarse la descripción simbólica (*assembler*) de una instrucción según:

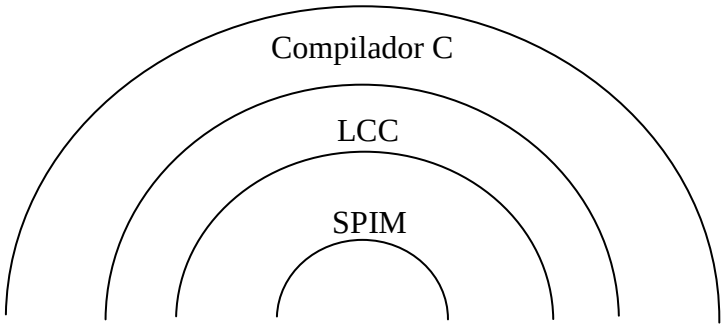
```
rótulo:     operación    op1, op2, op3        ; comentario de fin de línea.
```

Pueden agregarse, a los operandos de entrada, algunos símbolos especiales para describir el modo de direccionamiento. En el ejemplo siguiente, los paréntesis redondos implican un modo de direccionamiento.

```
rótulo:     operación    op1, offset(op2)
```

1.11. Aprender haciendo.

Es necesario acompañar este estudio con actividades prácticas, ya que mediante el apoyo de herramientas computacionales será más sencillo fijar los conceptos. Utilizaremos las siguientes herramientas:



Primero se experimenta con un compilador C, que se ejecute en un PC. Luego mediante el compilador cruzado LCC, se traduce a assembler. Posteriormente se visualiza en el simulador SPIM el lenguaje de máquina generado, y los datos.

Si bien se estudia un procesador determinado, el procedimiento de aprendizaje puede emplearse para otros procesadores, disponiendo de las herramientas adecuadas (compilador, simulador).

1.12. Ensambladores, Compiladores, Ligadores, Simuladores.

Se dispone de códigos binarios para las diferentes instrucciones de máquina. También existen códigos binarios para los números de los registros. Existen reglas para codificar los campos inmediatos de las instrucciones, tanto los campos de datos como los que establecen direcciones (en bifurcaciones y saltos).

La información anterior permite **ensamblar** instrucciones, se entiende por esto: pasar de assembler a código binario.

Cuestión que realiza eficientemente un programa ensamblador, pero que suele ser difícil empleando métodos de papel y lápiz. En los inicios de los computadores, los programas se escribían en binario, mediante conmutadores que permitían grabar cada bit de la instrucción, en una palabra de la memoria. El emplear mecanismos de abstracción para simbolizar los operandos y las direcciones fue un gran adelanto.

Al mismo tiempo se agregó al ensamblador la capacidad de incorporar constantes (equivalencias), y la de evaluar expresiones. De este modo algunos campos numéricos

pueden escribirse también en forma simbólica. (Desgraciadamente SPIM no dispone de esta característica). El ensamblador del microcontrolador 8051, dispone de esta facilidad.

Luego se le agregó simbolismos para programar la memoria de datos en: bytes, medias palabras, palabras, arreglos, strings, zonas inicializadas con valores establecidos, zonas de memoria reservadas para el programa no iniciadas con valores, lograr datos alineados. Permitiendo así manejar en forma simple las direcciones donde están las diferentes variables y constantes de un programa.

Después se agregó la posibilidad de crear secuencias de instrucciones y programarlas con un nombre abstracto. Esto se denomina **macro**. Este concepto no es una subrutina, sino un simple reemplazo de un texto (en una línea, la instrucción macro) por otro texto (generalmente en varias líneas, la expansión de la macro instrucción). Algunos ensambladores tienen macros predefinidas (es el caso de SPIM) para incorporar un repertorio ampliado; otros permiten que el programador cree su propia biblioteca de macros (es el caso del ensamblador para el 8051). En el manual del programador existe un listado completo de las macros que posee SPIM, su estudio permite ver cómo realizar nuevas acciones simples con las primitivas.

Más adelante se incorporó la posibilidad de disponer de bibliotecas de funciones o subrutinas, y de incorporar el código a un programa que las invoque. Ahora el programador debe conocer el prototipo de las funciones de la biblioteca y sus argumentos, para poder emplearlas. Se le adicionó programas para crear bibliotecas y mantenerlas (agregar, eliminar, y cambiar funciones). También un **ligador** (linker) para juntar las zonas de datos y funciones de biblioteca con las funciones y los datos del programa.

También se incorporó el ensamblado condicional, es decir que algunas partes del código se incorporen o no, dependiendo de alguna expresión o constante.

Gran parte de la historia del desarrollo del assembler fue heredada por el lenguaje C. Uno de los objetivos de su diseño fue precisamente reemplazar la programación assembler.

Un lenguaje de programación de alto nivel permite describir datos y algoritmos en forma abstracta. Un programa denominado compilador, compila un programa traduciéndolo al assembler de un determinado procesador (existe en la red el proyecto GNU, que permite obtener en forma pública diversos **compiladores** para procesadores específicos). Puede disponerse de un mismo programa en C, y compilarlo para procesadores diferentes.

Si el compilador se ejecuta en una máquina con otro procesador se dice que es un **compilador cruzado** (cross compiler) es el caso del lcc.
Un programa, ejecutándose en un determinado procesador, que permite ejecutar instrucciones de un procesador diferente se denomina **simulador**, es el caso de SPIM.

Índice general.

1. INTRODUCCIÓN. 1

1.1. REPERTORIO DE INSTRUCCIONES. 1
1.2. TIPOS DE REPERTORIOS. 2
1.2.1. Máquinas de Stack. *2*
1.2.2. Máquinas con acumulador. *3*
1.2.3. Máquinas de Registro-Memoria. *3*
1.2.4. Máquinas de Carga –Almacenamiento. (Load-Store) *4*
1.3. COMPARACIÓN DE LAS ARQUITECTURAS. 4
a) Máquina de stack: *4*
b) Máquina con acumulador: *5*
c) Máquina con operandos registro-memoria *5*
d) Máquina con registros carga-almacenamiento *5*
1.4. INSTRUCCIONES TÍPICAS DE UN REPERTORIO. 5
1.5. ESTADÍSTICAS DE USO DE INSTRUCCIONES. 7
1.6. MODOS DE DIRECCIONAMIENTO. 7
1.7. ESTADÍSTICA DE USO DE MODOS DE DIRECCIONAMIENTO. 9
1.8. ORGANIZACIÓN DE MÁQUINA. 9
1.9. PARTES DE UN COMPUTADOR. 10
1.10. NIVELES DE REPRESENTACIÓN. 10
1.10.1. Programación en lenguaje C. *11*
1.10.2. Programación assembler. *11*
1.11. APRENDER HACIENDO. 14
1.12. ENSAMBLADORES, COMPILADORES, LIGADORES, SIMULADORES. 14