Anexo A: Programación a bajo nivel de un computador

v 3.0.

Objetivo de la Práctica

- Ayudar a comprender el funcionamiento y la estructura de un computador Von Neuman mediante el estudio y la programación de un modelo sencillo.

Índice

1.- Lenguajes de bajo nivel

1.1.- Lenguajes de Programación

Un lenguaje de programación es un conjunto de símbolos y de reglas para combinarlos, que se usan para describir algoritmos. Los lenguajes de programación, al igual que los lenguajes que usamos para comunicarnos, poseen:

- **Léxico**: Es el vocabulario o conjunto de símbolos permitidos.
- **Sintaxis**: Indica como realizar construcciones del lenguaje.
- **Semántica**: Determina el significado de cada instrucción correcta.

Los lenguajes de programación, o lenguajes de alto nivel están específicamente diseñados para programar computadores. Sus características fundamentales son:

- Son independientes de la arquitectura física del computador. Por ello, no obligan al programador a conocer los detalles del computador que utiliza, y permiten utilizar los mismos programas en computadores diferentes. Existen incluso normalizaciones de lenguajes realizadas por organismos de estandarización (ANSI, ISO), las cuales rtatan de garantizar la portabilidad de los programas.

- Utilizan notaciones cercanas a las habituales en el ámbito en que se usan. Con estos lenguajes las operaciones se expresan con sentencias o frases muy parecidas al lenguaje matemático o al lenguaje natural (inglés).

1.2.- Lenguaje máquina

Como sabemos, los computadores son máquinas diseñadas para ejecutar programas. La ejecución de un programa exige que tanto el programa como los datos residan en memoria principal. La secuencia lógica de ejecución de cualquier programa se compone de los siguientes pasos, que se repiten sucesivamente:

1. Se **lee** de memoria principal una instrucción o sentencia del programa
2. La Unidad de Control **interpreta** la instrucción leída
3. Bajo las órdenes de la Unidad de Control se **ejecuta** la instrucción.

Sin embargo los computadores no son capaces de interpretar directamente un lenguaje de alto nivel, sólo son capaces de entender y ejecutar un lenguaje muy restringido, de bajo nivel, llamado **lenguaje máquina**. Por tanto se necesita un proceso de traducción de un programa escrito en un lenguaje de alto nivel a un programa en lenguaje máquina que sea posible de ser ejecutado por un computador.

Normalmente, una sentencia en un lenguaje de alto nivel da lugar, al ser traducida, a varias instrucciones en lenguaje máquina

El lenguaje máquina de cada computador depende estrechamente de su arquitectura interna. Existe, por tanto una incompatibilidad innata entre los distintos computadores o familias de computadores. Sin embargo , los fabricantes se esfuerzan por mantener una compatibilidad de código y que los programas máquina que funcionen en un procesador

sean capaces de funcionar en otro. Así los computadores de la serie Intel x86 son capaces de interpretar y de ejecutar todos el mismo conjunto de instrucciones.

El lenguaje máquina establece las capacidades básicas del computador, por lo que constituye una de las características más importantes de su arquitectura.

Las *instrucciones máquina* se almacenan y se tratan en el computador como cadenas de "*unos*" y "*ceros*". Se pueden escribir directamente en binario, o bien utilizar códigos intermedios como el *octal* o el *hexadecimal*. Como se puede uno imaginar esta es una forma muy engorrosa de escribir y manejar estas informaciones, por lo que la programación directa utilizando estas instrucciones requiere de técnicos especialistas en el propio hardware de la máquina.

1.2.1.- Campos de una instrucción máquina

Cada instrucción máquina contiene diversas informaciones o campos:

- **Código de operación**: especifica la operación que realiza la instrucción.
- **Campo de modo de direccionamiento**: Indica la forma en la que se expresan los operandos, aunque esto puede estar implícito en el código de operación.
- **Direcciones de los operandos**: Dependiendo de la operación, puede ser que en la instrucción se expliciten uno, dos o más operandos, cada uno de los cuales puede direccionarse de forma distinta.

Una instrucción máquina puede ocupar una o varias palabras de memoria, dependiendo de los operandos que tenga y de los modos de direccionamiento para esos operandos.

1.2.2.- Propiedades de las instrucciones máquina

Las **propiedades** generales que suelen cumplir las instrucciones de máquina son las siguientes:

- **Realizan una única y sencilla función**, por lo que su decodificación e interpretación es muy sencilla.
- **Emplean un número fijo de operandos con una representación determinada**: No deben tener una sintaxis ambigua.
- **La codificación de las instrucciones es bastante sistemática**, puesto que ello facilita su decodificación. Siguen el mismo patrón.
- **Las instrucciones son autocontenidas e independientes**. Es decir, contienen toda la información necesaria para ejecutarse, no requiriendo de la información de otras instrucciones y no dependiendo su interpretación de la posición que ocupan en la memoria o en el programa.

1.2.3.- Definición de un juego de instrucciones

La **definición de un juego de instrucciones** requiere detallar los siguientes aspectos:

- **Operaciones realizadas**: Qué operaciones realiza.
- **Representación o representaciones de los datos**: Cómo se especifican los datos sobre los que operan las instrunciones.
- **Modos de direccionamiento**: Mecanismos por los que se puede especificar un operando o la especificación de una instrucción.
- **Formato o formatos de las instrucciones**: Modo en que se codifican las instrucciones.

1.2.4.- Modos de direccionamiento

Un modo de direccionamiento es *un procedimiento que permite determinar un operando, o la ubicación de un operando o de una instrucción.*

Aquello que queremos direccionar puede residir en la propia instrucción, en un registro o en la memoria principal, siendo e objetivo de los modos de direccionamiento especificar el lugar concreto donde se encuentra.

Los modos de direccionamiento más usuales:

- **Direccionamiento implícito**: Corresponde a situaciones en las que el operando o su dirección se encuentran en un registro, y éste se encuentra implícitamente especificado en el propio código de operación de la instrucción.

- **Direccionamiento inmediato**: El operando forma parte de la propia instrucción como dato.

- **Direccionamiento directo**: La instrucción contiene la dirección de memoria o del registro en la que se encuentra el operando.

- **Direccionamiento indirecto**: En la instrucción se indica el lugar en donde se encuentra la dirección del operando.

- **Direccionamiento indexado**: Interviene un registro índice en el que se almacena un desplazamiento. La instrucción contiene una dirección de referencia y la dirección del operando se obtiene sumando a esa dirección de referencia al valor contenido en el registro índice.

- **Direccionamientos relativos**: La dirección del operando se obtiene añadiendo a una dirección de referencia un desplazamiento que se da en la propia instrucción como dirección. Dependiendo de dónde esté ubicada esa dirección de referencia se definen los tipos de direccionamiento relativo:
  - Direccionamiento relativo a base.
  - Direccionamiento relativo a Contador de Programa.
  - Direccionamiento relativo a Segmento.

1.2.5.- Tipos de instrucciones

El juego de instrucciones de un computador debe cumplir dos condiciones:

- **Debe ser completo**: Con él se puede calcular, en un tiempo finito, cualquier tarea computable.

- **Debe ser eficaz**: Ha de permitir una alta velocidad de cálculo, sin exigir a cambio una alta complicación de la Unidad de Control ni de la Unidad Aritmética.

Desde el punto de vista de los computadores comerciales, lo que tiene interés es que sean rápidos y económicos, por lo que habrá de dotarles de un conjunto bien seleccionado de instrucciones. La selección del juego de instrucciones de un computador es, por tanto, uno de los puntos más críticos de su diseño.

Las instrucciones las podemos clasificar en 4 grupos:

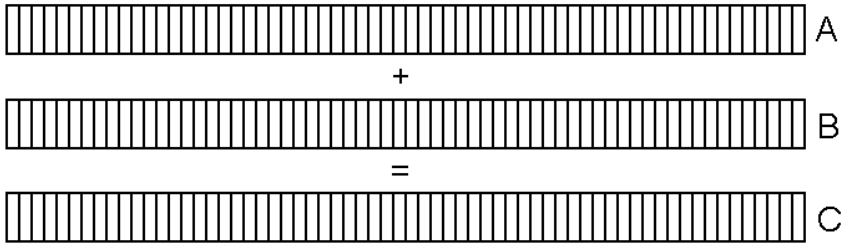
- **Transferencia de información**
- **Aritmético-Lógicas y de desplazamientos de bits**.
- **Trasnferencias de control**: Saltos, llamadas a procedimientos, Parada del procesador,...
- **Misceláneas**: Entrada-Salida, Temporizaciones, ...

1.3.- Instrucciones Vectoriales y Multimedia

Para acelerar la ejecución de las instrucciones aritméticas se han diseñado dos alternativas que se basan en que una misma instrucción realice la misma operación sobre un conjunto de datos. estas dos alternativas son las instrucciones vectoriales y multimedia.

1.3.1.- Instrucciones vectoriales

Los supercomputadores vectoriales se caracterizan porque tienen formatos de operación vectorial, es decir, que están compuestos por **n** operandos escalares. Además, dispone de instrucciones vectoriales que afectan a todos los elementos del vector. Por ejemplo, con una instrucción vectorial de suma se ejecutan **n** operaciones suma de manera que se suman las **n** parejas de elementos de dos registros vectoriales, dejándose el resultado en otro registro vectorial.



Suma Vectorial

La ganancia de velocidad se consigue por dos razones. Por un lado, hay que leer menos instrucciones de la memoria principal y, por otro lado, la unidad aritmética de estas máquinas está preparada para ejecutar este tipo de instrucciones lo que permite ejecutar muy rápidamente las n operaciones relativas a una instrucción vectorial.

1.3.2.- Instrucciones multimedia

A mediados de los años 90 se inicia la explosión de las aplicaciones multimedia, con el tratamiento generalizado del sonido y de las imágenes. Estas aplicaciones exigen una gran cantidad de procesamiento matemático y precisan de unos formatos de datos muy específicos. Así, el sonido se digitaliza con 16 bits de precisión, por lo que exige un tratamiento de entero con signo de 16 bits, por otra parte, cada píxel que forma parte de una imagen RGB requiere tres palabras de 8 bits que almacenan sendos enteros positivos.

En un computador convencional de 32 bits de ancho de palabra, por cada muestra de sonido procesada se está aprovechando solamente 16 de esos 32 bits de la palabra. En la imágenes la situación aún es peor, puesto que se ha de procesar cada byte de forma independiente.

Para mejorar esta situación los fabricantes de computadores han introducido unas nuevas instrucciones que se denominan de multimedia.

Estas instrucciones se basan en dos principios comunes al procesamiento multimedia y de comunicaciones:

- El procesamiento del sonido, de las imágenes y de las comunicaciones exige **repetir** las operaciones aritméticas y lógicas sobre un gran número de operandos de pequeño tamaño (típicamente 16 bits para el sonido y 8 bits para las imágenes).

- Los algoritmos empleados suelen exigir una **aritmética de saturación** en la cual, si un resultado produce desbordamiento, se toma el máximo valor representable como solución.

Las **características de las instrucciones multimedia** son las siguientes:

- **Operan sobre varios datos simultáneamente**: La palabra del computador se divide en varios operandos independientes, realizándose la operación de forma independiente sobre cada dato. Por ejemplo, se divide la palabra de 32 bits en 4 palabras de 8 bits y se almacena un dato en cada una.

- **Se incluyen diversas instrucciones que operan sobre datos de 8 ó de 16 bits**.

- **Existen instrucciones especiales** que utilizan aritmética con saturación y sin signo, y otras que utilizan aritmética con saturación y signo.

- La ventaja de estas instrucciones es que permiten un **paralelismo en el tratamiento de los datos** con el consiguiente ahorro de tiempo. Para permitir este paralelismo hay que complicar ligeramente el diseño de la Unidad Aritmética, de manera que pueda trabajar con 1, 4 u 8 unidades independientes.

1.3.3.- Instrucciones MMX de Intel

El **juego de instrucciones MMX** de Intel se incorporó en su Pentium en Marzo de 1996

Como hemos visto para mejorar la multimedia y el procesado 3-D, la nueva tecnología MMX de Intel puede empaquetar múltiples pixeles en un registro del procesador y manipularlos con una sola instrucción.

Para conseguir una mayor compatibilidad y modificar lo mínimo posible sus procesadores en lugar de añadir registros físicos nuevos a la arquitectura x86 Intel reutiliza los registros coma flotante (FP) existente a modo de registros lógicos MMX. Las instrucciones MMX utilizan sólo la porción de mantisa de 64 bits de los registros FP de 80 bits, ignorando la porción de 16 bits del exponente. Esto da lugar a 8 registros lógicos de 64 bits sin que se altere de forma significativa la arquitectura x86. Los nuevos procesadores x86 que soporten MMX serán capaces de direccionar los nuevos registros, desde el **MM0** hasta el **MM7.**

Las instrucciones MMX pueden empaquetar varios tipos de datos dentro de estos registros de 64 bits:

- **Bytes** empaquetados (ocho por registro)
- **Palabras** empaquetadas (cuatro por registro);
- **Palabras dobles** empaquetadas (dos por registro)
- **Palabra cuádruple** (un valor de 64 bits por registro).

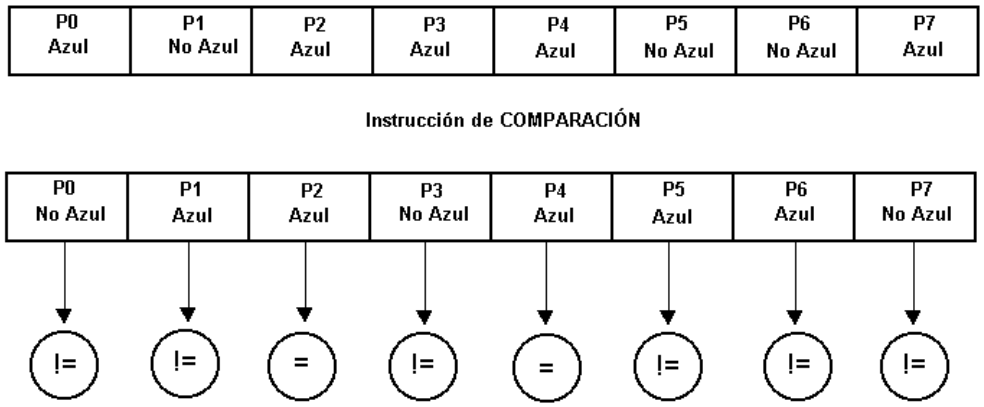
Estos tipos de datos resultan útiles debido a que, normalmente, los programas multimedia trabajan con pequeñas unidades de datos.

Ejemplo: Un píxel de color en modo *color real* , la resolución de color más utilizada, usa 24 bits: 1 byte por cada color RGB. Este modo permite trabajar con hasta 16,7 millones de colores, mucho más de lo que es capaz de distinguir el ojo humano.

El inconveniente de esta técnica reside en la imposibilidad de que los programadores puedan entremezclar instrucciones PF y MMX, ya que necesitan los mismos registros. Pero esto no es tan importante como parece, ya que los programas multimedia efectúan habitualmente sus operaciones en coma flotante antes de visualizar los datos (el proceso de modelado se basa fundamentalmente en instrucciones para enteros).

MMX presenta un conjunto de instrucciones para enteros, de propósito general, que utiliza el **paradigma SIMD** (*Single Instruction/Multiple Data*). Una sola instrucción procesa todos los datos de los registros empaquetados. Este paralelismo incrementa el rendimiento. Casualmente este concepto no es nuevo para Intel. Hace años, la familia **i860RISC**, ahora obsoleta por completo, ofrecía una **tecnología** parecida.

Ejemplo: Instrucción de comparación de pixels.



Rendimiento en velocidad del paralelismo

En la ejecución de esta instrucción podemos apreciar que:

1. Las instrucciones MMX pueden empaquetar valores de 6, 16, 32 o 64 bits dentro de registros MMX de 64 bits. Aquí 8 pixeles de 8 bits cada uno son empaquetados en un solo registro MMX.

2. Cuando se ejecuta una instrucción MMX, los ocho valores de píxel de cada registro MMX se procesan simultáneamente. Aquí, la instrucción compara los píxels de los registros 1 y 2 en busca de igualdad.

3. Resultados de las comparaciones. El paralelismo que se logra cuando una única instrucción MMX procesa múltiples valores de píxel resulta más

rápido que si tuvieran que ejecutar instrucciones separadas para cada comparación.

Debido al éxito comercial y en a rendimiento que ha tenido esta nueva tecnología, los procesadores de AMD han incorporado su tecnología **3DNow!**, que consiste en un juego de instrucciones similares a as MMX y, posteriormente el mismo Intel introdujo otro nuevo juego de instrucciones multimedia (**MMX2**) en sus procesadores Pentium II con las que se cubría un mayor abanico de aplicaciones además de incluir en la fabricación del procesador nuevos registros específicos para tratarlas.

Para escoger este nuevo conjunto de instrucciones Intel formó un grupo de arquitectos de chips y expertos en algoritmos para analizar vídeo interactivo, realidad virtual y aplicaciones 3D de alta calidad, y encontramos que estas aplicaciones aparentemente diferentes tienen mucho en común. Tienen grandes cantidades de computación, ejecución en paralelo, y tienden a usar datos de tipo '*small integer*'. Estas nuevas instrucciones, usando una técnica SIMD, fueron escogidas específicamente para mejorar los algoritmos del núcleo y mejorar así el rendimiento global de estas aplicaciones.

Vista de los procesadores Intel, a ala Izquierda y AMD a la derecha

2.- Lenguajes ensambladores

El lenguaje máquina de cualquier computador tiene la notable ventaja de que es directamente utilizable por éste , sin necesidad alguna de transformación previa. Normalmente, este uso inmediato lleva consigo un alto rendimiento del computador en cuanto a:

- **Volumen de memoria principal utilizada**: Se utiliza mucho menos memoria.
- **Longitud de los programas resultantes**: Los programas son más cortos.
- **Velocidad de ejecución**: Estos programas se ejecutan mucho más rápidos.

Como inconvenientes de la programación en lenguaje máquina tenemos que:

- Las instrucciones en lenguaje máquina se han de dar en binario, en octal o en hexadecimal, lo cual hace que no sea muy legible a ojos humanos.

- El programador ha de realizar la asignación de memoria, decidiendo en qué posiciones coloca las diferentes unidades de que consta su programa, las variables y los datos.

Para solucionar estos dos inconvenientes, se utiliza el **lenguaje ensamblador**. En un programa en ensamblador cada instrucción viene identificada mediante un *nombre indicativo* o *mnemotécnico* de la acción a realizar por la instrucción, de manera que un programa en lenguaje ensamblador no es directamente ejecutable por el computador, requiriendo una traducción que lleve de los nombres simbólicos a los códigos y direcciones de máquina correspondientes, pero sí que es entendible por un programador humano.

En los lenguajes ensambladores cada nombre simbólico de instrucción corresponde a una instrucción en código máquina, de manera que la traducción es muy sencilla e inmediata y se utiliza el propio computador para ejecutar un *programa traductor*, que se conoce con el nombre de **ensamblador**. De esta forma, un **traductor ensamblador** es un programa que recibe como datos de entrada una serie de instrucciones (un programa) en lenguaje ensamblador y produce como resultado de salida una serie de instrucciones en lenguaje máquina (cadena de *ceros* y *unos*), existiendo una correspondencia en cuanto al tipo y número de ambas series de instrucciones.

Al escribir un programa en lenguaje ensamblador se utilizan direcciones simbólicas y se utilizan también nombres simbólicos para las variables. Por tanto, utilizando un lenguaje ensamblador, se obvian los inconvenientes señalados para el lenguaje máquina, sin perder ninguna de sus ventajas, dado que a cada instrucción en lenguaje ensamblador le corresponde una instrucción máquina.

De esta forma, estos lenguajes permiten al programador:

- Escribir las instrucciones utilizando para representar los códigos de operación una **notación simbólica o mnemotécnica** en lugar de códigos numéricos. Normalmente estos códigos simbólicos están constituidos por unas pocas letras que, en forma

abreviada indican la operación a realizar. Usualmente, debido al origen de los fabricantes de computadores, los nemotécnicos son abreviaturas en inglés.

- Utilizar **direcciones simbólicas** de memoria, a modo de variables, en lugar de direcciones binarias absolutas. existen sentencias declarativas para indicar al traductor la correspondencia entre direcciones simbólicas y direcciones de memoria. Con estas pseudoinstrucciones, el traductor crea una tabla con cuya ayuda, al generar las instrucciones máquina, sustituye las direcciones simbólicas por las direcciones binarias correspondientes.

- Es posible **insertar líneas de comentarios** entre las líneas de instrucciones. el traductor las elimina automáticamente, no incluyéndolas en el código máquina que genera.

3.- Arquitectura y programación de un computador sencillo

3.1.- Arquitectura del computador

Al igual que los datos, los programas se representan dentro del computador mediante bits.

Un programa está constituido por instrucciones

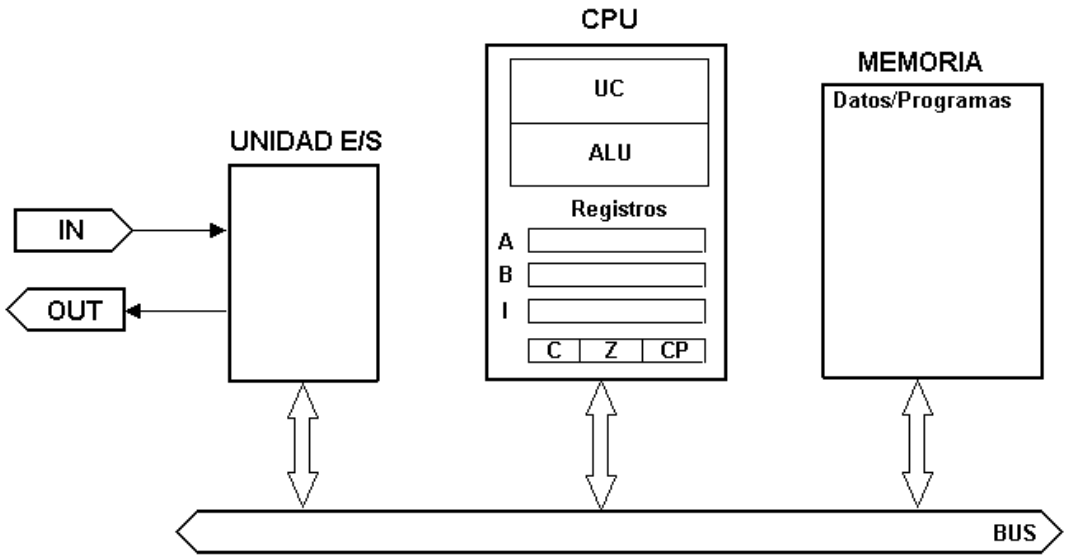
- Una instrucción especifica una operación definida al procesador.
- Las instrucciones se codifican según el formato de instrucciones asociado a cada procesador.

Como hemos visto el formato de instrucción está constituido normalmente por los campos:

- **Código de operación**: Indica el tipo de instrucción.
- **Campos de direccionamiento**: Indican la forma de obtener los operandos y de almacenar el resultado.

Cada fabricante de procesadores impone su propio juego de instrucciones que tiene que proporcionar a los fabricantes de compiladores para que elaboren el código máquina apropiado.

Vamos a estudiar como máquina ejemplo la arquitectura y programación de un computador muy sencillo:



Podemos observar que se trata de un computador que sigue la arquitectura de **Von Neuman.**

Está construido bajo una arquitectura de **10 bits**. Tanto el **bus de datos** y el **bus de direcciones** son de **8 bits**. Lo cual permite direccionar hasta 2^{10} (1024) posiciones de memoria distintas y representar hasta 2^{10} (1024) datos distintos.

La representación de los números se hace en signo y magnitud. Con 108 bits, el rango de números representables es de : [-1023,+1023].

3.2.- Juego de Instrucciones

El juego de instrucciones del que disponemos es:

INSTRUCCIONES DE TRANSFERENCIA

Instrucción	Significado
(1) MOVE operando1, operando2	Carga el operando 2 en el operando 1, cuando el operando 1 es un registro o una posición de memoria.

Dependiendo del tipo de direccionamiento, así funcionará la instrucción. Uno de los dos operandos debe ser un registro.

Ejemplo:

MOVE A, [Dirección]
Carga en el registro A el valor almacenado en la dirección especificada en la instrucción

MOVE [Dirección], B
Carga el contenido de B en la dirección de memoria especificada en la instrucción

MOVE #valor, A
Error, no está permitida esta operación.

MOVE [Dirección], [Dirección]
Error, no está permitida esta operación.

INSTRUCCIONES ARITMÉTICAS

Instrucción	Significado
(2) ADD Operando1, Operando2	Carga en el registro A la suma del operando1 más el operando2

Uno de los dos operandos debe ser un registro:

Ejemplo:

ADD B, #15
Carga en A el contenido de B más el valor 15
ADD [10], A
Carga en A el contenido de la dirección de memoria 10 más el contenido de A
ADD [10], #15
Error, no está permitida esta operación

Instrucción	Significado
(3) SUBTRAC Operando1, Operando2	Carga en el registro A la resta del operando1 menos el operando2

Uno de los dos operandos debe ser un registro:

Ejemplo:

SUBTRAC B, #15
Carga en A el contenido de B menos el valor 15
SUBTRAC [10], A
Carga en A el contenido de la dirección de memoria 10 menos el valor del registro A
SUBTRAC [10], #25
Error, no está permitida esta operación

Instrucción	Significado
(4) INCREMENT registro	Incrementa en 1 el valor del registro indicado. El registro puede ser A ó B

Instrucción	Significado
(5) MUL Operando1, Operando2	Carga en el registro A el producto del operando1 por el operando2

Uno de los dos operandos debe ser un registro:

Ejemplo:

MUL A, #15
Carga en A el contenido de A por el valor 15
MUL [10], B
Carga en A el contenido de la dirección de memoria 10 por el valor del registro B
MUL [10], #25
Error, no está permitida esta operación

Instrucción	Significado

(6) DIV Operando1, Operando2	Realiza la división del operando1 entre el operando2 y Carga en el registro A el cociente y en el registro B el resto
Uno de los dos operandos debe ser un registro: Ejemplo: **DIV B, #15** Carga en A el contenido de B dividido entre 15 y en B el resto **DIV [10], A** Carga en A el contenido de la dirección de memoria 10 dividida entre A y en B el resto **DIV [10], #25** Error, no está permitida esta operación	

Si como consecuencia de una operación aritmética se produce desbordamiento, el resultado puede ser impredecible.

INSTRUCCIONES DE COMUNICACIÓN

Instrucción	Significado
(7) PORT Puerto, Registro	Si el puerto es de entrada (IN) carga en el registro indicado el contenido de ese puerto Si es de salida (OUT) carga en el puerto el contenido del registro indicado
Los puertos se van a representar con la entrada y la salida estandar de la máquina: Puerto **IN**: Teclado Puerto **OUT**: Pantalla	

INSTRUCCIONES DE CONTROL

Instrucción	Significado
(8) STOP	Detiene el funcionamiento del procesador
Dentro de un programa se pueden escribir tantas instrucciones STOP como se quiera, el procesador se detendrá al encontrar la primera instrucción.	

Instrucción	Significado
(9) COMPARE registro, Operando	Compara el contenido del registro indicado con el operando indicado, estableciendo los indicadores **C** y **Z** de la máquina

La forma de establecer los indicadores:

- Si Registro > Operando → C=0 y Z=0
- Si Registro < Operando → C=1 y Z=0
- Si Registro == Operando → C=0 y Z=1

INSTRUCCIONES DE SALTO

Instrucción	Significado
(A) JUMP TO Dirección	Carga en el CP la dirección de memoria especificada en la instrucción

Instrucción	Significado
(B) JUMP IF LESS THAN Dirección	Almacena en el CP la dirección especificada en la instrucción si **C=1**

Instrucción	Significado
(C) JUMP IF GREATER THAN Dirección	Almacena en el CP la dirección especificada en la instrucción si **C=0**

Instrucción	Significado
(D) JUMP IF EQUAL TO Dirección	Almacena en el PC la dirección especificada en la instrucción si **Z=1**

Instrucción	Significado
(E) JUMP IF NOT EQUAL TO Dirección	Almacena en el PC la dirección especificada en la instrucción si **Z=0**

Este juego de 14 instrucciones forma un conjunto completo con el que se puede construir cualquier programa estructurado.

3.3.- Modos de Direccionamiento

A la hora de representar los datos de las instrucciones, el programador puede utilizar cualquiera de estas 3 alternativas:

- **Direccionamiento inmediato**

  Se indica el valor tal cual. Debe ir precedido del símbolo #

  Ejemplo: MOVE A, #23
  Mueve el valor 23 al registro A

- **Direccionamiento directo a registro:**

  Se indica el nombre del registro con el que se opera:

  Ejemplo: INCREMENT A

- **Direccionamiento directo a memoria**

  El dato al que se hace referencia se encuentra en la dirección indicada: [Dirección]

  Ejemplo: MOVE A, [23]
  Mueve el contenido de la dirección de memoria 23 al registro A

- **Direccionamiento indirecto**

  El dato al que se hace referencia se encuentra en la dirección de memoria indicada por el registro I: [I]

  Ejemplo: MOVE A, [I]
  Mueve el contenido de la dirección de memoria indicada por I al registro A

3.4.- Programación con HMJ

Los programas escritos en ensamblador HMJ-8 deben seguir las siguientes reglas para que sean ejecutados por la máquina:

- Deberán estar formados por una lista de instrucciones separadas por saltos de línea (una instrucción debajo de otra) en un fichero de tipo texto con extensión **hmj**.
- La primera instrucción se almacena en la dirección de memoria 0, el contenido inicial de los registros, de la memoria y de los indicadores es 0.
- Al principio de cada instrucción se deberá escribir el número de línea de esa instrucción. Los números de línea no tienen porqué ser correlativos, pero si que deben formar una sucesión monótona estrictamente creciente.
- Los comentarios se indican con "`//`" y se considera comentario todo lo escrito desde ahí hasta el final de la línea.

3.5.- Ejemplos de Programas

Ejemplo 1

Almacenar en A el valor de 8

```
0  MOVE A, #8
1  STOP
```

Ejemplo 2

Almacena el valor 67 en la posición de memoria 10

```
0  MOVE A, #67
1  MOVE I, #10
2  MOVE [I], A
3  STOP
```

Ejemplo 3

Almacena en A el resultado de 20 + 15 – 5

```
0  MOVE A, #15
1  SUBTRACT A, #5          //A:=15-5
2  ADD A, #20              //A:=20+10
3  STOP
```

Ejemplo 4

Se lee un dato de entrada si el valor coincide con el valor almacenado en la dirección de memoria 10 entonces se visualiza dicho valor por el puerto de salida.

```
0     PORT IN, A
1     COMPARE A, [10]
2     JUMP IF NOT EQUAL TO 5
3     PORT OUT, A
4     STOP
```

Ejemplo 5

Mostrar por el puerto de salida la serie numérica de 0 hasta un número dado por el puerto de entrada.

```
0  PORT IN, A       //Lee el valor del puerto y lo almacena en el registro A
1  MOVE [164], A    //Almacenamos en la dirección de memoria 164 el valor leído
2  COMPARE A, #0
3  JUMP IF LESS THAN 11 //Si el número que nos dan es menor que 0, no visualizamos nada
4  MOVE A, #0
5  PORT OUT, A
6  INCREMENT A
7  MOVE B,[164]
8  COMPARE B, A
9  JUMP IF LESS THAN 11
10 JUMP TO 5
11 STOP
```

Ejemplo 6

Mostrar por el puerto de salida la tabla de multiplicar del 7.

```
0  MOVE B, #7
1  MOVE [200], B
2  MOVE B, #0
3  MUL B, [200]
4  PORT OUT A
5  INCREMENT B
6  COMPARE B, #11
7  JUMP IF LESS THAN 3
8  STOP
```

Ejemplo 7

Dados dos números por el puerto de entrada, mostrarlos por orden creciente en el puerto de salida.

```
0  PORT IN, A
1  PORT IN, B
2  COMPARE A, B
3  JUMP IF GREATER THAN 7
4  PORT OUT A
5  PORT OUT B
6  JUMP TO 9
7  PORT OUT B
8  PORT OUT A
9  STOP
```

Ejemplo 8

Mostrar por el puerto de salida la serie de números de Fibonacci hasta un número indicado en el puerto de entrada.

```
0  PORT IN, A
1  MOVE [200], A      //almaceno el número de elementos de la serie que tengo que mostrar
2  MOVE A, #0
3  MOVE [201],A       //almaceno el número de elementos de la serie que llevo mostrados
4  COMPARE A, [200]
5  JUMP IF LESS THAN 7
6  JUMP 30
7  MOVE A, #1
8  PORT OUT A         //Muestra 1
9  MOVE [201],A
10 COMPARE A, [200]
11 JUMP IF LESS THAN 13
12 JUMP TO 31
13 MOVE B, #1
14 PORT OUT A         //Muestra 1
15 MOVE [202], A      //salvo el valor de A
16 INCREMENT A
17 MOVE [201], A
18 COMPARE A, [200]
19 JUMP IF LESS THAN 21
20 JUMP TO 31
21 MOVE A, [202]
22 ADD A, B
23 MOVE B, [202]
24 PORT OUT A         //Muestra el siguiente elemento de la serie
25 MOVE [202], A
26 MOVE A, [201]
27 INCREMENT A
28 MOVE [201], A
29 COMPARE A, [200]
30 JUMP IF LESS THAN 21
31 STOP
```

3.6.- Programa compilador de ficheros HMJ

El programa funciona bajo el Sistema Operativo MS-DOS y se ejecuta :

```
C:\> HMJ-10.exe
```

Su funcionamiento es el de emular un entorno de operación sobre la máquina HMJ-8.

```
hmj>_
```

Sobre ese entorno se pueden ejecutar programas **hmj** y comprobar el valor de los registros, los indicadores y las posiciones de memoria.

Las ordenes que reconoce y ejecuta el programa son:

- **salir**: sale del programa
- **inicio** : inicializa valores en registros, índices y memoria
- **reg A:** Muestra el valor del registro A
- **reg B:** Muestra el valor del registro B
- **reg PC:** Muestra el valor del registro PC
- **ind C:** Muestra el valor del indicador C
- **ind Z:** Muestra el valor del indicador Z
- **mem xx** : Muestra el valor de la posición de memoria **xx**
- **ejec nombre.hmj** : Ejecuta el fichero nombre.hmj

Para depurar un programa se puede parar la ejecución en cualquier punto y comprobar el valor de los registros, los indicadores y la memoria.

Ejemplo:

```
hmj>ejec p1.hmj

Ejecutando programa: p1.hmj
in>10

out>1
out>1
out>2
out>3
out>5
out>8
out>13
out>21
out>34
out>55
<fin>
hmj>
```

4.- Ejercicios

Programa 1 (Práctica Junio 2001)**:**

Implementar un programa que lea por el puerto de entrada un número entero positivo **n** y muestre por el puerto de salida un **1** si el número es primo y un **0** si no lo es.

Programa 2 (Práctica Junio 2001)**:**

Implementar un programa que lea por el puerto de entrada **n** números enteros y los muestre por el puerto de salida ordenados de menor a mayor. El valor de **n** debe ser leído por la máquina.

Programa 3 (Práctica Junio 2000)**:**

Implementar un programa que lea por el puerto de entrada un número **n** y muestre por el puerto de salida, los **n** primeros números primos.

Programa 4 (Práctica Junio 2000)**:**

Implementar un programa que lea por el puerto de entrada dos números enteros positivos: **(a, b)** y muestre por el puerto de salida su *Máximo Común Divisor* (**mcd**).

Programa 4 (Práctica Septiembre 2000)**:**

Implementar un programa que lea por el puerto de entrada dos números enteros positivos: **(a, b)** y muestre por el puerto de salida su *Mínimo Común Múltiplo* (**mcm**).

Programa 5 (Examen Diciembre 2000)**:**

Realizar los siguientes programas utilizando las instrucciones del computador HMJ-8:

- *Leer 3 números y mostrarlos ordenados de menor a mayor.*
- *Leer dos números a, b y mostrar por la salida* a^b.

Programa 6 (Examen Junio 2000)**:**

Realiza el siguiente programa utilizando las instrucciones del computador HMJ-8: *Dado un número **n** por el puerto de entrada, mostrar 1 si es primo y 0 si no lo es*.

Programa 7 (Examen Junio 2001)**:**

El logaritmo por defecto en base n de un número X es el mayor entero p tal que $n^p \leq X$. Por ejemplo, el logaritmo por defecto en base 10 del número 9 es 0, el de 85 es 1, el de 277 es 2, etc, ...

Diseñar un programa en ensamblador del computador sencillo HMJ-10 que reciba como parámetros sendos números n y X y calcule el logaritmo por defecto en base n del número X. Los números n y X son enteros positivos de 10 bits representados en signo y magnitud, y se encuentran almacenados en los registros A y B respectivamente al comenzar la ejecución. El programa devolverá el valor calculado por el puerto de salida. Suponer que X>0, y que siempre se cumplirá que $n^p<2^{10}$, o sea, que n^p siempre cabra en un número de 10 bits.