5 Lenguaje de Máquina

Ya visto el detalle de la organización de la CPU y el nivel de microprogramación, en este punto veremos las consideraciones del nivel de lenguaje de máquina. Este conjunto de instrucciones son las interpretadas por la CPU, es decir son las que caracterizan un programa ejecutable. En este punto estudiaremos su clasificación en términos de la operación que implementan y la manera en que se direccionan sus argumentos. Para fijar estos conceptos en un lenguaje concreto mostraremos como ejemplo el formato de las instrucciones de la arquitectura MIPS (arquitectura RISC) utilizadas por fabricantes como NEC, Nintendo, Sony, Sylicon Graphics en el desarrollo de sus arquitecturas de CPU.

Razón de ser de la existencia de lenguaje de máquina

Este nivel es la interfaz entre el compilador- ensamblador y el intérprete (microprograma) La pregunta a plantear entonces es: ¿por qué el compilador - ensamblador no genera directamente código en micro-instrucciones?. Algunas de las justificaciones de diseño vienen dadas por:

a) Implica una mayor complejidad en el desarrollo del compilador, pues tendría que transformar de manera óptima cada sentencia de un lenguaje de alto nivel en un conjunto de instrucciones con muchas restricciones (poco flexibles y estrechamente condicionadas a la funcionalidad y organización de hardware)

b) Dado que las micro-instrucciones son directamente interpretadas por el hardware, su trama de bits generalmente no está comprimida, para evitar introducir dispositivos decodificadores - que hacen más costoso y lento el hardware-. Por lo tanto es un hecho que una micro-instrucción contiene más bits que una instrucción de máquina y adicionalmente cada instrucción de máquina se descompone en muchas micro-instrucciones. Eso implica que el espacio de memoria para almacenar un programa como conjunto de micro instrucciones crece en promedio en un orden de magnitud. Y como la eficiencia de ejecución de una instrucción está condicionada en gran medida al tiempo de acceso a memoria (***instruction fetch***), esto redunda en un peor desempeño de ejecución, junto con un uso mayor de recursos de memoria.

Pese a estas consideraciones de diseño que justifican la necesidad del nivel de micro-programación, existe una línea de diseñadores de arquitecturas de CPU que han eliminado este nivel haciendo un diseño de lenguaje de máquina más simplificado, de tal forma que cada instrucción de máquina corresponda a un ciclo de máquina (trayectoria de datos). Estas arquitecturas son llamadas de **formato de instrucciones reducido** (su sigla en ingles es **RISC)**. Se ha mostrado que para determinados procesos de mucha carga aritmética real, estas arquitecturas muestran mejores prestaciones (aplicaciones gráficas, y algoritmos con aritmética real pesada). No obstante y pese a esto, los computadores de propósito general (convencionales) típicamente son micro-programados, por las razones antes mencionadas.

Independiente de la existencia de arquitecturas micro-programadas el nivel de lenguaje de máquina (o el nivel relevante para un programador de nivel 2) se puede entender como el interpretado por la CPU, independiente si dichas instrucciones están asociadas o no a las señales de control de una trayectoria de datos.

5.1 Formato de Instrucciones de Máquina

En este punto se verán los aspectos del formato de instrucciones, considerando sus tipos de campos y algunos aspectos de diseño. Hay dos aspectos importantes en la codificación de una instrucción, el relacionado con el tipo de instrucción y el relacionado con los modos de direccionamiento de sus argumentos.

"*El objetivo del diseño de instrucciones es proveer facilidad, flexibilidad y eficiencia a las operaciones que de forma más frecuente requieren las aplicaciones de alto nivel* "

Consideraciones de Diseño

En la determinación del formato de las instrucciones (relacionado estrechamente con el diseño de la arquitectura de la CPU), deben considerarse muchos aspectos:

- El tamaño de la instrucción: mientras menos bits ésta tenga será más eficiente su almacenamiento en memoria y su acceso. Esto pues un cuello de botella importante en la velocidad de ejecución de instrucciones es la tasa o número de instrucciones por segundo que se pueden transferir desde la memoria principal a la CPU. Aquí esta implícito el ancho de banda que provee el bus y el tiempo promedio de acceso a memoria. De esta manera el diseño de las instrucciones incide en el desempeño del sistema.

- Restricciones de largo: Otro aspecto de diseño es que el bus de datos por ser líneas paralelas, transfiere palabras de 8, 16, 32 bits. Correspondientemente la memoria permite direccionar el mismo tipo de palabras, por tanto para aprovechar íntegramente todos los bits de la codificación de una instrucción, ésta debe ser un múltiplo de dicha palabra.

- Codificación de la Operación: Los bits del ***opcode*** deben permitir identificar de manera única todas las posibles instrucciones básicas que tiene este nivel, es decir el mínimo número de bits que posee es $\log_2(N)$, con N el número de operaciones a implementar por la arquitectura.

- Bits de direccionamiento: Los bits de direccionamiento deben ser capaces de indexar todo los bloques de almacenamiento provistos, ya sea los registros internos de la CPU, la memoria principal o los puertos de I/O.

- Argumentos: Otro aspecto es que dependiendo del tipo de instrucción es el número de argumentos que ésta tendrá y por tanto puede variar el formato de los restantes campos que la componen. Esto implica la existencia de instrucciones de largo variable.

Formato MIPS (RISC)

En algunos diseños el formato de una instrucción depende solamente del tipo de operación, información almacenada en el ***opcode***. Es decir el campo ***opcode*** define también el número y formato de los restantes campos.

Analizaremos un ejemplo del conjunto de instrucciones de máquina de una arquitectura MIPS de 32 bits y 32 registros. El campo ***opcode*** en esta arquitectura tiene tamaño fijo de 6 bits (posibilita 2^6 tipos de instrucciones). En este caso determina tres tipos de formatos de instrucciones que describiremos en lo que sigue. Sus instrucciones de máquina independiente de su formato son de tamaño fijo, 32 bit, lo que facilita el diseño de la lógica de control para implementar la operación de ***Instruction Fetch***. [1]

R-format (Operaciones Aritméticas y Lógicas)

Corresponde al formato de instrucciones utilizado para implementar las operaciones aritmética-lógicas (las que utilizan las funcionalidades de la ALU en la organización). El formato de su trama de bits se ve en la siguiente figura:

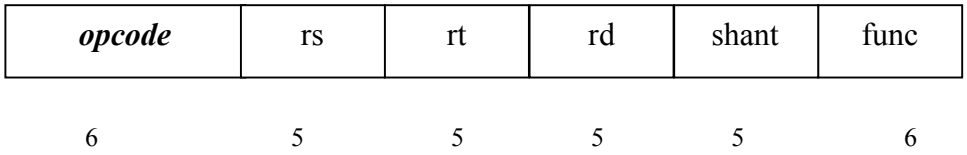


Figura lm.9: *Campo del formato de una instrucción aritmético (R-format)*

Observaciones

- Este tipo de instrucciones tienen tres argumentos dos fuentes y uno de destino. Los 5 bits sólo permiten direccionar los 32 registros de la CPU. Es decir en esta arquitectura no se permite operaciones aritméticas sobre posiciones de memoria en una instrucción de lenguaje de máquina.

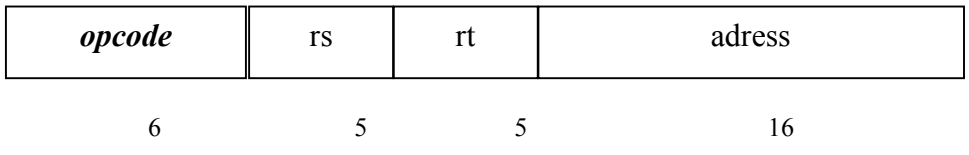
- ***func*** es un campo adicional para determinar específicamente la operación aritmética a ejecutar, es decir una expansión implícita del ***opcode***.

La sintaxis en assembler de este tipo de instrucciones es:

add s1,s2,s3 // s1=s2+s3
sub s1,s2,s3 // s1= s2-s3

S-format

Este tipo de instrucciones tienen en sus campos información para direccionar la memoria principal. El campo ***adress*** en codificación binaria con signo permite direccionar (ya sea para lectura o escritura) un conjunto de posiciones en un rango de $+-2^{15}$ respecto al contenido de un registro interno de direcciones codificado por **rt.**



[1] En el capitulo de pipeline, se vera que el diseño del lenguaje de máquina de esta arquitectura (RISC) facilita la incorporación de las técnicas de pipeline, lo que hace más eficiente el proceso de ejecución de instrucciones.

Figura lm.10: *Formato de instrucciones de movimiento de datos (I-format)*

La sintaxis del assembler de este tipo de instrucciones es:

lw s1,100(s2) // s1=Memoria (100+s2)
sw s1,100(s2) // Memoria(100+s2)=s1

Este formato de instrucciones tiene un diseño asociado a su naturaleza, implica dos argumentos, fuente y destino. En esta arquitectura (que tiene un espacio de direcciones de 32 bits y consecuentemente puede direccionar 2^{32} bytes) la dirección es obtenida por medio de un registro de dirección más una constante de 16 bits codificada como parte de la instrucción.

J-format

Este tipo de instrucción tiene como objetivo modificar el contenido del PC y por consiguiente afectar el flujo de ejecución de un proceso. Notar que del punto de vista de su formato no tiene argumentos, el registro destino es implícito (la instrucción modifica el PC). En el campo ***address*** se encuentra la información de que valor asignar al PC (codificación entera sin signo)

opcode	adress
6	26

Figura lm.11: *Formato de Instrucción de salto incondicional* ***(J- format)***

La sintaxis del assembler es:

j 1000 // PC = 1000

La constante 1000 queda codificada en el campo ***address*** de la instrucción

Observación :

Del formato de las operaciones aritméticas y lógicas, se puede decir que estas sólo actúan sobre los registros. Existen otros diseños de CPU en los cuales sus operaciones aritméticas permiten considerar como argumentos tanto registros como posiciones de memoria principal (arquitecturas típicamente micro-programadas). Ejemplo (intel 8088, 80286 y DP11)[2].

[2] **Tanenbaum** "Organización de Computadores un enfoque estructurado" ejemplos de formato de instrucciones Cap 5.1

5.2 Direccionamiento

La manera de accesar los datos de entrada salida esta ligado al diseño del nivel de lenguaje de máquina y la determinación del formato de las instrucciones. Dependiendo de la arquitectura existen instrucciones con tres direcciones, dos fuentes y un destino. Por ejemplo la arquitectura MIPS (RISC) mencionada anteriormente permite operaciones del tipo:

$$destino = fuente_1 + fuente_2$$

Hay otras arquitecturas de CPU (arquitecturas 8088,80286,80386,DPD-11) que sólo permiten dos argumentos (fuente y un destino), de esta manera la operación anterior se vería como:

$$destino = destino + fuente$$

En este ejemplo particular una instrucción de máquina en la primera arquitectura se traduce en dos instrucciones de la otra. Esto no quiere decir que una arquitectura ejecute dicho proceso más rápido que la otra, pues esto al final depende de los ciclos de máquina en los cuales se traducen dichas sentencias y el número de lectura para extraer las instrucciones de memoria.

Se recapitularán los modos de direccionamiento más importantes presentes en la mayoría de las arquitecturas.

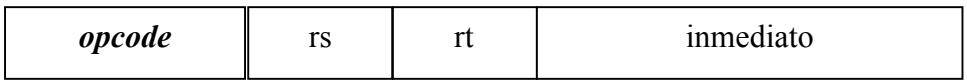
Direccionamiento Inmediato

Este tipo de direccionamiento tiene de manera explícita como campo de su formato un operando de dicha instrucción. Es decir el argumento es extraído directamente del **IR** y por consiguiente no se necesita accesar ni los registros de la CPU, ni la memoria principal. La ventaja de este modo de direccionamiento es que no se requiere un acceso adicional a memoria para cargar un operando de la instrucción.

Ejemplo

Una instrucción de máquina de la arquitectura MIPS con direccionamiento directo es:

addi *sp,sp,4 // sp=sp+4*



6 5 5 16

El formato de esta instrucción es equivalente a la mostrada en esta arquitectura para las operaciones de trasferencia de datos *(S-format*), pero los 16 bit del campo de direcciones almacenan el argumento de manera inmediata (es decir puede representar constantes enteras entre $+-2^{15}$)

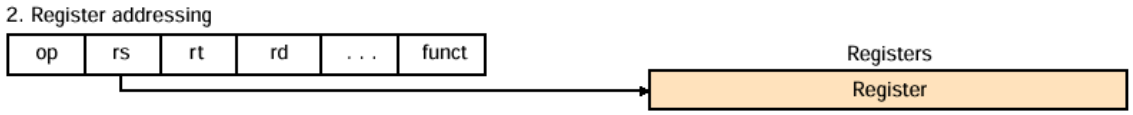
Direccionamiento Directo

En este modo de direccionamiento un campo de la instrucción tiene la información de la posición de memoria donde esta el bloque o palabra binaria a considerar como argumento de la instrucción. Naturalmente el tamaño de dicho campo debe estar asociado a los bits que utiliza la arquitectura para direccionar cada uno de los bloques (bytes o palabras) que posee la memoria principal. Por

ejemplo con 16 bit de direcciones se pueden direccionar 64KB y con 32 bit se direccionan 4GB. Este tipo de direccionamiento en algunas arquitecturas se trata de evitar pues implica una instrucción de tamaño comparativamente mayor, lo cual redunda en varias lecturas desde la memoria de programa.

Direccionamiento Directo por Registro

Es conceptualmente equivalente al direccionamiento directo, pero en este caso el campo contiene los bits para direccionar el espacio de memoria de registros de la CPU, lo que implica una codificación significativamente más pequeña. Este es una de los modos de direccionamiento presentes en todas las arquitecturas.



Ejemplo MIPS:

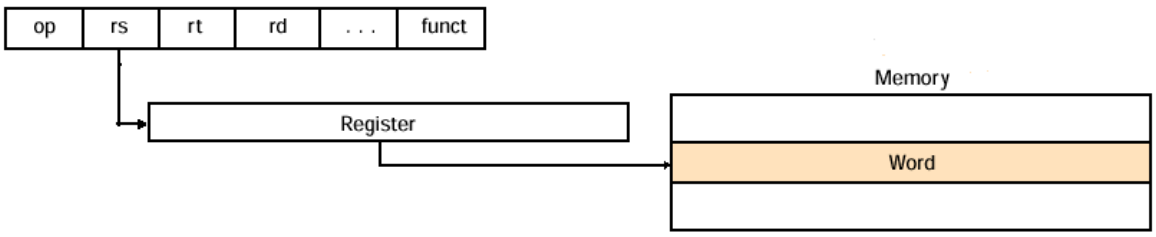
add s1,s2,s3 // s1=s2+s3

opcode	rs	rt	rd	shant	func
6	5	5	5	5	6

Los campos rs, rt, rd utilizan 5 bit para direccionar uno de los 32 registros de la CPU.

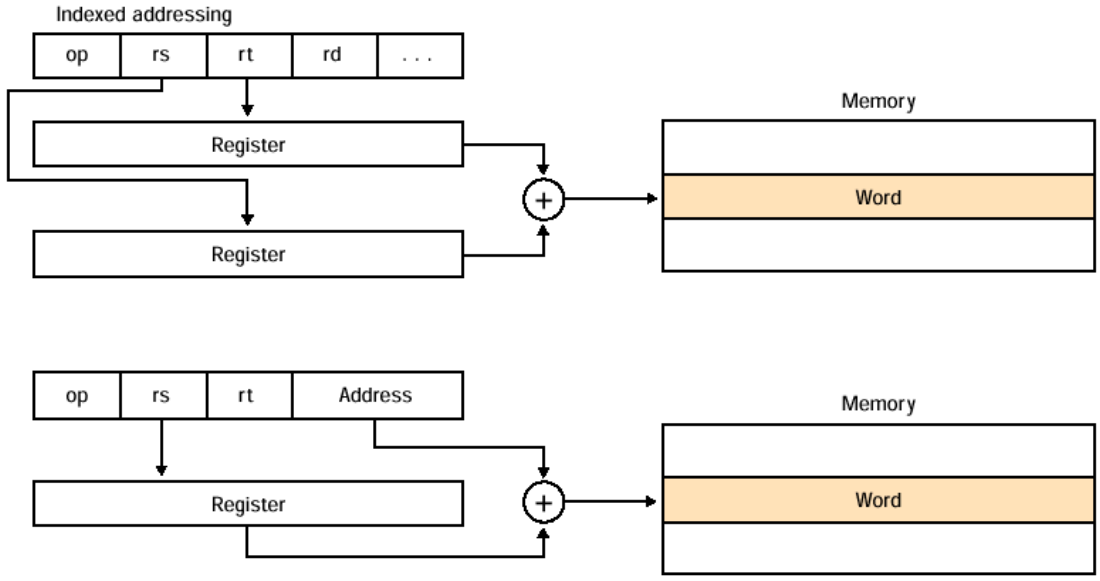
Direccionamiento Indirecto

En este formato los campos del argumento direccionan la palabra en memoria o en el espacio de registros de la CPU que contiene la dirección de memoria del operando. Este tipo de direccionamiento implica hacer 2 o más accesos a memoria. En otras palabras, la instrucción direcciona un puntero a la palabra a considerar como argumento de la instrucción (de entrada o salida). Existen en algunas arquitecturas registros especiales para direccionar la memoria.



Direccionamiento Indexado(Aritmética para el cálculo de direcciones**)**

En este formato el direccionamiento a una posición de memoria corresponde a la suma de una dirección base presente en un campo de la instrucción o en un registro (**registro base**), más el contenido de un registro de índice (**offset**). Todos estos registros son parte de la codificación del argumento.



Esta metodología permite accesar elementos de un arreglo en memoria contigua de manera eficiente en conocimiento de la dirección base de dicho arreglo. También se utiliza para accesar a campos de estructuras. Incluso hay arquitecturas que diseñan este tipo de direccionamiento de tal manera que al accesar la posición de memoria dada por el registro índice más la base, paralelamente incrementan el registro índice, para un potencial acceso a la siguiente posición de memoria. Esta técnica es llamada de ***auto-indexación***.

Clasificación del lenguaje según la implementación de sus Modos de Direccionamiento

Un aspecto importante en el diseño de las instrucciones es de que manera incorporar los distintos modos de direccionamiento. Se tienen dos opciones:

- RISC: se codifica como parte del **opcode** tanto la operación como la información de los modos de direccionamiento de sus argumentos. Por lo tanto es el ***opcode*** quien unívocamente determina el formato restante de la instrucción (ejemplo mostrado de la arquitectura MIPS)

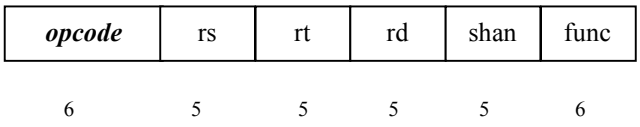
- **Arquitecturas Ortogonales**: El ***opcode*** sólo codifica el tipo de operación entre sus argumentos, permitiendo que los campos asociados a los argumentos determinen el tipo de direccionamiento (sub-campo llamado ***modo***) y según éste la interpretación de los bits restantes asociado a dicho argumento. En este esquema el sub-campo modo de los bits del argumento determina el formato restante de la **instrucción** (ejemplo DPD-11). De esta manera cualquier operación permite todos los modos de direccionamiento que soporte la arquitectura.

Ejemplo de la operación suma con distintos modos de direccionamiento en arquitectura MIPS

Direccionamiento por registro

add s1,s2,s3 // s1=s2+s3

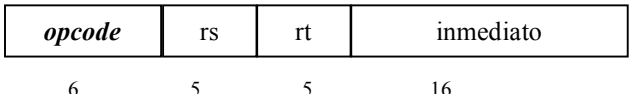
Formato Instrucción de máquina



Direccionamiento por registro e inmediato

addi *sp,sp,4 // sp=sp+4*

Formato Instrucción de máquina



Utilidad típica de los modos de direccionamiento para el manejo de variables de alto nivel

Directo: acceso a variables globales

Inmediato: movimiento de constantes

Registros: acceso interno a variables temporales que maneja la CPU (compilador)

Registros indirectos: apuntadores a segmentos o estructuras en memoria.

Indexado: acceso a variables locales (segmento de pila), acceso a elementos de arreglo en memoria contigua, acceso a campos de estructura.

Auto indexado: apilar y desapilar parámetros de procedimiento

Ejemplo de la sintaxis del Assembler del 80386 y sus modos de direccionamiento

Mov Ebx, Eax: instrucción que transfiere los bit del registro fuente (Eax) al destino (Ebx), lo que implica un direccionamiento por registro.

Add Al,5: esta instrucción suma 5 al registro acumulador Al, utiliza direccionamiento inmediato para almacenar la constante y direccionamiento de registro para el acumulador.

Or Al, [100H] esta instrucción hace un Or lógico entre el acumulador AL y la posición de memoria 100 en hexadecimal, utilizando un modo de direccionamiento directo a memoria (como parte de la codificación de la instrucción esta la posición de memoria del argumento fuente)

Mov Eax, [Ecx] instrucción que mueve al registro Eax (direccionado de manera directa) el contenido de memoria direccionado por el registro ***Ecx***, utilizando para éste un esquema de direccionamiento indirecto por registro.

Mov Al, 100H[Esi] mueve al registro acumulador Al (direccionado de manera directa), el contenido de la posición de memoria, resultado de la suma del registro base ***Esi*** mas la constante de desplazamiento 100H, utilizando un modo de direccionamiento indexado.

Mov Eax, 5[Ebx+Esi] equivalente a la anterior pero el direccionamiento del argumento fuente, corresponde a la posición de memoria dada por la suma de un registro base Ebx más la suma de un registro índice más la constante de desplazamiento 5 (direccionamiento indexado por registro base y desplazamiento)

5.3 Clasificación de Instrucciones

En esta clasificación se incorporan los tipos de instrucción más comunes presentes en las arquitecturas.

Instrucciones de Movimiento de datos

Corresponde a la transferencia de información de un bloque de memoria a otro, estos bloques pueden ser registros o celdas en memoria principal Estas instrucciones tienen distinta naturaleza dependiendo de la manera en que direccionan sus argumentos y de la unidad de memoria a transferir (byte, palabra (2bytes), doble palabra (4bytes), etc.)

Las sintaxis más comúnmente encontrada son (***move***, ***load***, ***store, push, pop***)

Operaciones Binarias

Corresponden a las operaciones que combinan sus datos de entrada para generar un resultado que se almacena en un operando destino. En esta clasificación se encuentran:

- Operaciones aritméticas (suma y resta de números enteros)
- Multiplicación y división de números enteros
- Operaciones booleanas (AND, OR, OR exclusivo)
- Operaciones aritméticas de punto flotante o punto fijo
- Comparaciones

Este tipo de operaciones modifica el registro de estado del proceso **STATUS**, que almacena información de control de la última operación realizada por la ALU (zero, overflow, set, carry_out, etc)

Operaciones Unarias

Son operaciones que sólo poseen un operando, por lo tanto eso implica que dicho argumento actúa como entrada y salida de la operación.

- Operaciones de desplazamiento (shift) izquierda - derecha (utilidad aritmética)
- Operaciones de rotación (donde el bit que se inserta en la palabra producto de la rotación es el que se eliminó en el otro extremo de ella).
- Inicializar (poner en cero el argumento seleccionado)
- Incrementar, decrementar (típicamente sobre codificación entera)

5.4 Aspectos de manejo de Memoria (Stack)

En este nivel de lenguaje de máquina se deben dar las herramientas necesarias para facilitar las tareas que realizan los lenguajes de alto nivel. Estos lenguajes están diseñados en estructuras de bloque, es decir implementan funciones y procedimientos para resolver tareas específicas. Cada uno de estos bloques utiliza temporalmente un espacio de memoria para sus variables locales y una vez terminado el conjunto de instrucciones que lo componen retornan un resultado luego del cual debe ser liberado el espacio de memoria utilizado.
Mediante la administración de una pila de ejecución (**stack**) el nivel de lenguaje de máquina da solución a este problema, que incide tanto en el flujo en que se ejecuta el programa (PC), como en la administración de la memoria para datos

Una pila es un segmento en la memoria principal, en la cual se manejan las variables locales en la llamada a procedimientos y también se guardar el estado del proceso previo a la llamada. Este segmento esta controlado por un registro interno de la CPU conocido por el **stack pointer** (SP)

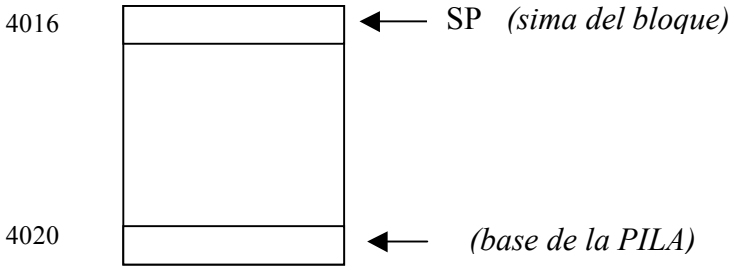


Figura lm.1: *Bloque de memoria principal administrado por el SP utilizando el concepto de PILA.*

Se distinguen varias operaciones básicas presentes en el ámbito del lenguaje de máquina, con las cuales se puede administrar el contenido de la pila, como por ejemplo la instrucción ***PUSH* X** que apila el contenido de un registro X y luego decrementa el ***SP*** . Otra función importante es la instrucción ***POP Y*** que des-apila el elemento apuntado por el **SP** y se incrementa **SP**.

La PILA permite hacer el proceso de asignación de espacio de memoria tanto para los argumentos de la llamada a un procedimiento, como sus variables locales. Al mismo tiempo permite almacenar la dirección de retorno de la siguiente instrucción a ejecutar después de terminado el procedimiento, por lo tanto se utiliza para almacenar el estado del proceso previo – a la llamada de un procedimiento. .,

Al llamar a un procedimiento ocurren los siguientes eventos:

- El **PC** salta a la dirección donde está almacenado el conjunto de instrucciones que componen el procedimiento, luego se apila la dirección de retorno y el estado de los registros internos de la CPU (*puede existir una instrucción especial de llamada a procedimiento* ***Call*** *que su decodificación implica todas las transferencias señaladas*). También se apilan los argumentos del procedimiento de alto nivel (instrucciones de máquina **PUSH X**).

- Se pide espacio en la pila para las variables locales del procedimiento, es decir se decrementa el SP tantos bytes como variables locales posea el procedimiento. (la pila crece hacia direcciones de memoria más baja)

- Al terminar la rutina se libera el espacio utilizado por las estructuras y variables locales (incrementando el SP) se des-apila la dirección de retorno (POP PC) y finalmente se incrementa el SP tantas veces como argumentos haya tenido el procedimiento.

- Con el nuevo valor del PC sigue el flujo de ejecución del proceso.

Observación

Aquí se evidencia la manera en que a nivel de lenguaje de máquina se direccionan las variables locales de un procedimiento. Esto no es por un direccionamiento absoluto (que dependería del segmento de memoria asignado a la PILA) si no que por un direccionamiento relativo al SP, que es independiente de cual es el estado de SP previo a la llamada del procedimiento.

Del punto de vista de la memoria de programa, las instrucciones que componen un proceso típicamente tienen un orden correlativo en memoria asociado a la secuencia en que van a ser ejecutadas. Las excepciones son las instrucciones que modifican el flujo de ejecución del proceso. Una de esas instrucciones es la instrucción que llaman a procedimientos (**call** *address*), pues los procedimientos típicamente no se encuentran en la trama de memoria contigua que componen el programa principal (**main**). De esta forma el cambio del flujo se traduce en la modificación del PC con la dirección que es parte de la información que codifica este tipo de instrucción.

El siguiente ejemplo muestra como se administra una PILA cuando a un procedimiento es llamado.

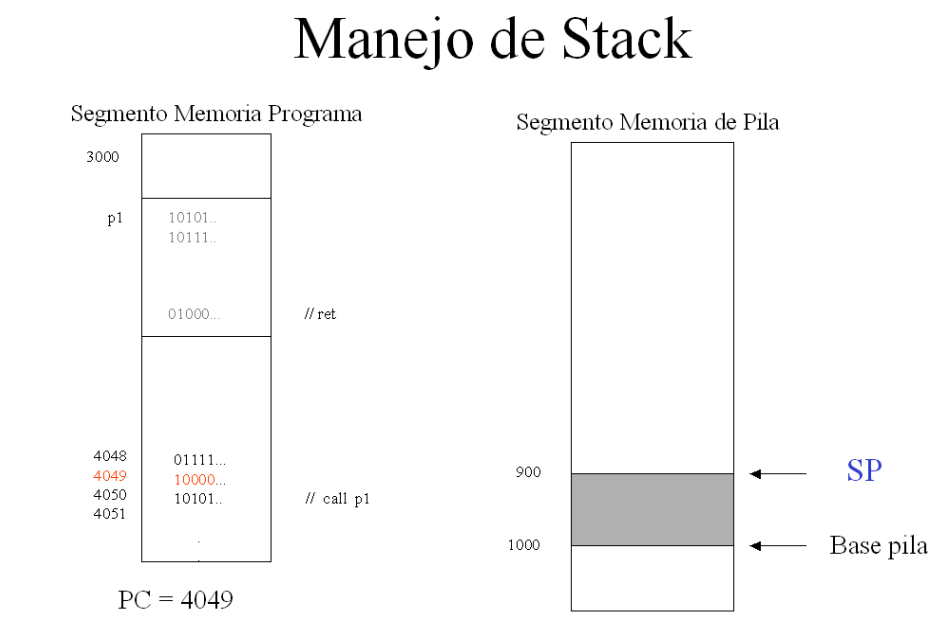


Figura lm.2: *Estado antes de la llamada a un procedimiento. Se distingue el segmento de memoria de programa de la rutina principal, el procedimiento y la memoria de dato administrada por la pila El PC apunta a la instrucción previa a la llamada del procedimiento.*

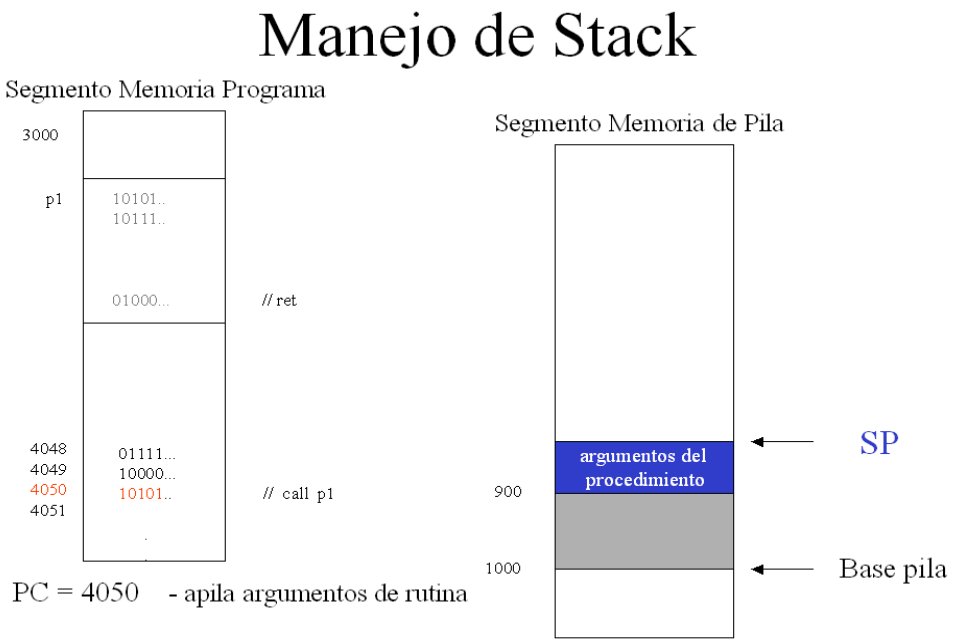


Figura lm.3: *Ejecución de la llamada al procedimiento p, fase apilar los argumentos*

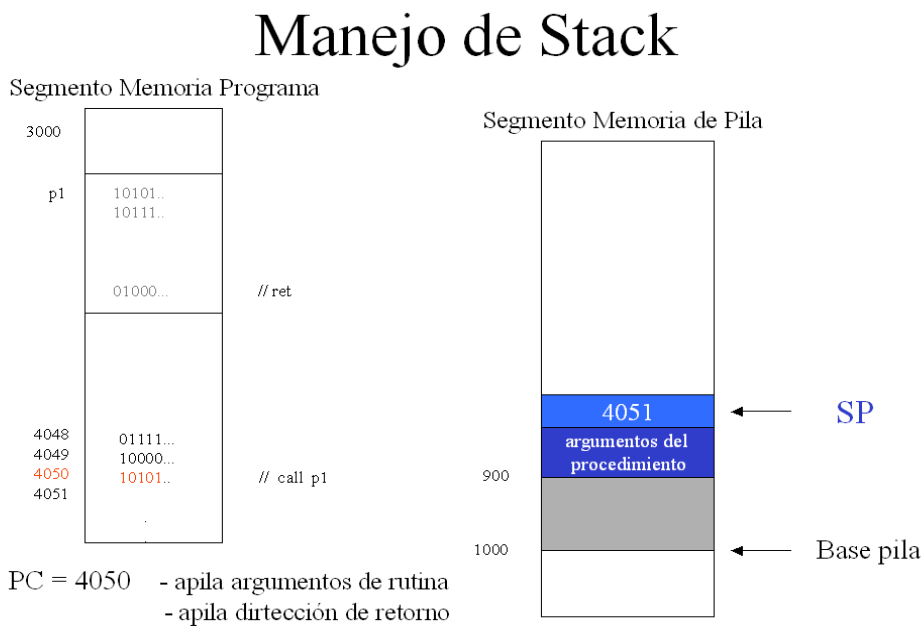


Figura lm.4: *Ejecución de la llamada al procedimiento p, fase apilar dirección de retorno*

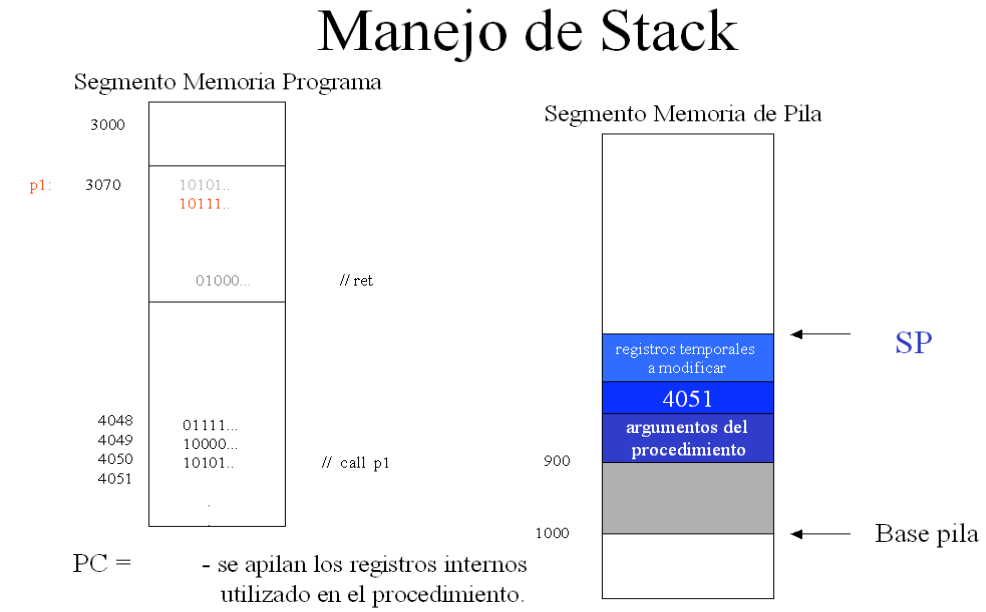


Figura lm.5: *Se modifica el PC y se apila el estado de los registros de la CPU (almacena el estado de la CPU)*

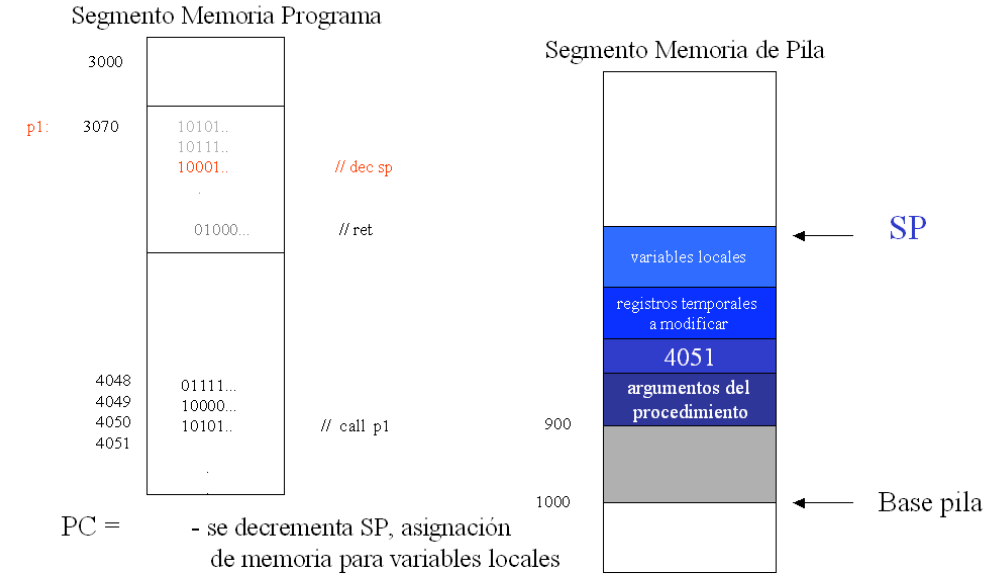


Figura lm.6: *Asignación de espacio en el segmento de pila para variables locales definidas en el procedimiento.*

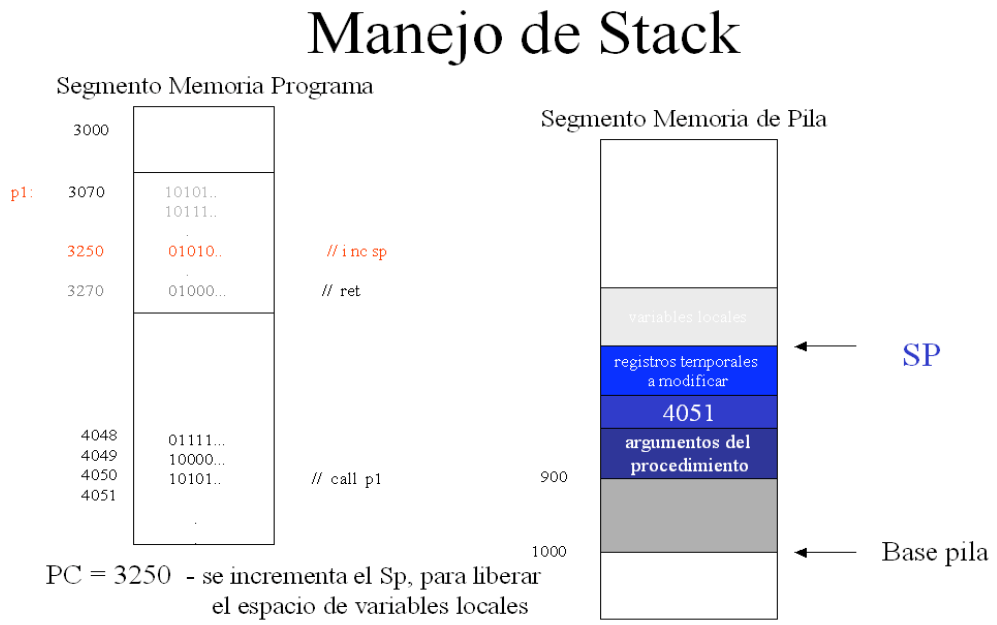


Figura lm.7: *Después de la última instrucción valida del procedimiento, vienen las instrucciones de liberación de la memoria de las variables locales.*

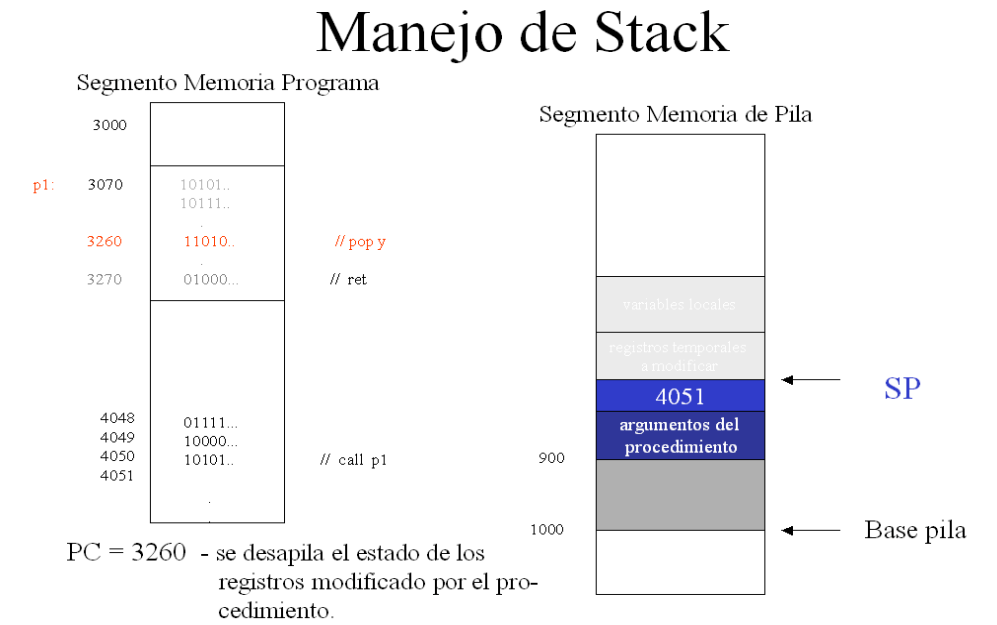


Figura lm.8: *Des –apilar los valores de los registros de la CPU previa a la llamada*

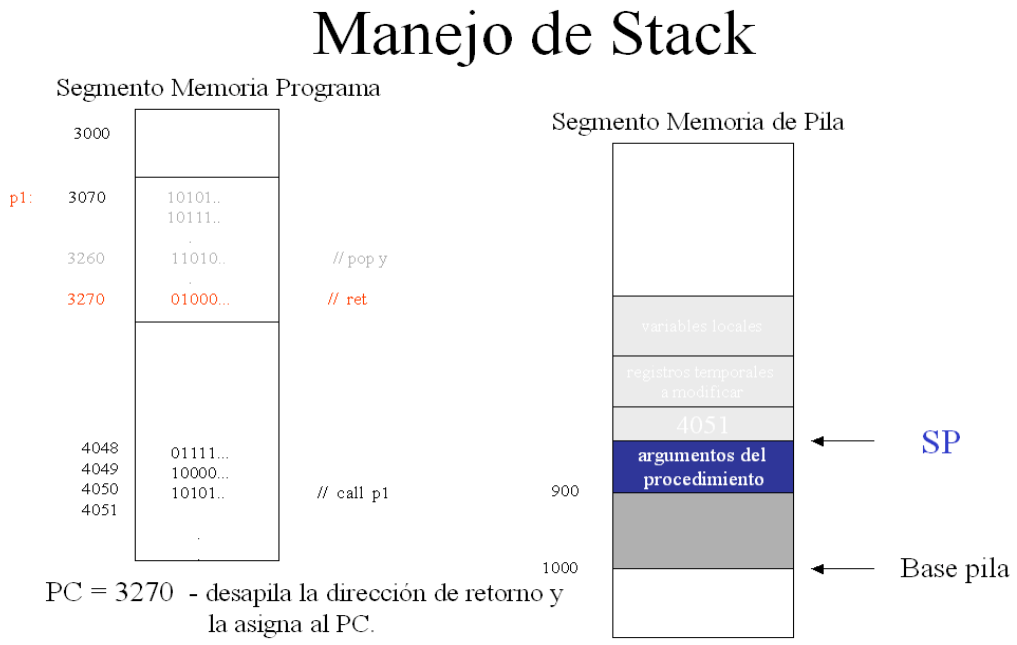


Figura lm.9: *Des –apilar el valor del PC antes de la llamada y modificar con el este el flujo de ejecución del proceso.*

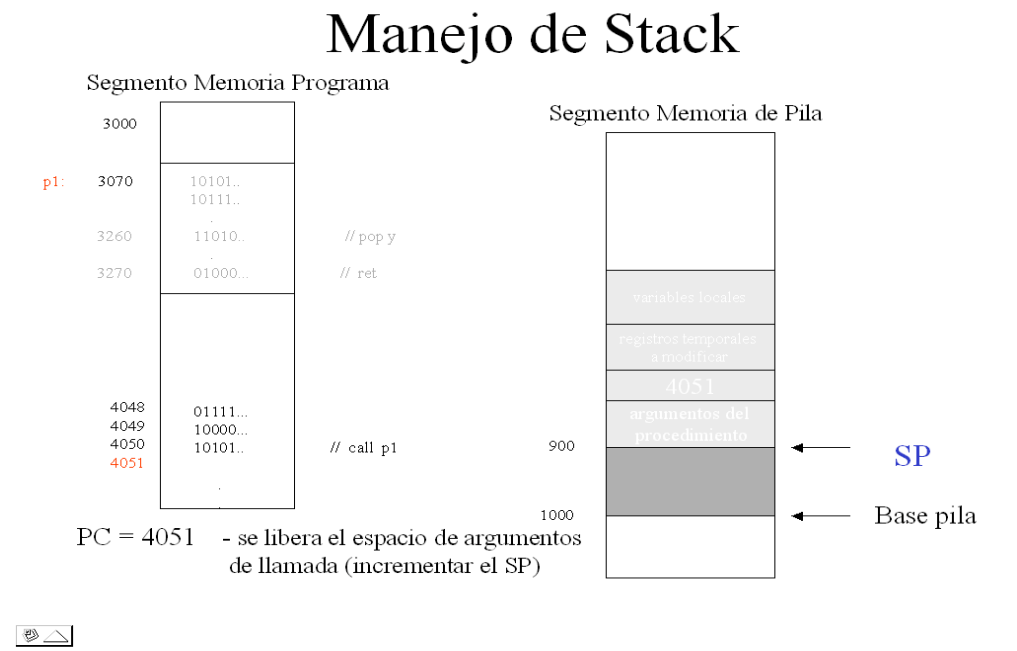


Figura lm.9: *Incrementar el SP para liberación de los argumentos pasados por valor al procedimiento*

Cada una de estas fases de asignar, apilar, liberar y des-apilar información del **stack,** implican transferencias de información, las cuales están asociadas a instrucciones del tipo (call, pop, push o return) en la mayoría de las sintaxis de lenguajes de máquina.

5.5 Instrucciones Asociadas al flujo de Ejecución de Procesos

i) Saltos condicionales

Este tipo de instrucciones permiten manejar al nivel de lenguaje de máquina parte del flujo de ejecución del proceso. Corresponden a las instrucciones en lenguaje de máquina que implementan sentencias de alto nivel del tipo ***if, while,*** and ***for*** , donde el flujo del programa cambia como función de alguna condición sobre el estado del proceso (almacenado en un registro de la CPU *STATUS)*. Estas instrucciones típicamente sondean el valor de algún bit determinado de dicho registro y en termino de ello generan saltos en el programa.

ii) Llamadas a Procedimientos

Vimos que los procedimientos son el eslabón de los programas estructurados, y que a nivel de lenguaje de máquina se pueden manejar mediante la administración de un segmento de pila (SP), tanto para la asignación y direccionamiento de sus argumentos y variables (variables locales), como para almacenar la información de parte del estado del proceso justo antes de su llamada (PC, otros registros de propósito específico)

Al nivel de lenguaje de máquina las arquitecturas pueden tener instrucciones básicas para el manejo de la pila (***Pop x, Push y***), como también instrucciones de llamada a procedimientos ***call*** y retorno de procedimiento ***return***, que modifican el PC y hacen el manejo de la pila para guardar o recuperar el estado del proceso.

iii) Interrupciones

Son cambios del flujo del programa (modificación del PC) no ocasionados de manera directa por instrucciones propias de éste (Jump, call, ret), si no por eventos externos asociados a los dispositivos de I/0.

Análogo a los procedimientos estos eventos modifican el flujo lógico del programa de manera asíncrona, no ajustado a necesidades propias del proceso, sino a una necesidad externa de mayor prioridad.

Las interrupciones son una de las maneras de atender las solicitudes de transferencia de información de los dispositivos de I/O y por tanto implican suspender la ejecución del proceso actual de menor prioridad y ejecutar las rutinas de atención de interrupciones, ubicadas en otro espacio de la memoria de programa.

Los eventos que se suceden al momento de ocurrir una interrupción son:

1 Apilar la dirección de retorno en un segmento de pila especial de interrupciones (dependiendo de la arquitectura puede ser el mimo).

2 Modificar el PC hacia una posición fija de memoria donde se encuentra la rutina de atención de interrupciones. En este punto dicha rutina identifica el dispositivo y direcciona la PC al espacio de memoria específico donde se encuentra **su rutina de atención**, utilizando una tabla o vector de interrupciones.

3 La rutina de interrupciones salva todos los registro a modificar (apila en un segmento de pila especial de interrupciones), para posterior recuperación del estado del proceso interrumpido.

4 Ejecutar el código de atención de interrupciones (acceso a los registros de dispositivos de I / O, ya sea para lectura o escritura)

5 Restaurar del estado del proceso (registros internos de la CPU)

6 Actualizar del PC, para recuperar el estado del flujo del proceso original.

De esta manera nuevamente el ***stack*** es la entidad encargada de dar transparencia a las interrupciones, de tal forma que el flujo de ejecución y el estado del proceso interrumpido no sea afectado en su correctitud.

A nivel del hardware de la CPU, este una línea especial para la llamada de interrupciones, a la cual todos los dispositivos I/O se encuentran físicamente conectados. Para incorporar prioridades de interrupción existen esquemas de conexión o dispositivos que administran dichas prioridades. En el caso de la CPU Intel 8088 dicho controlador de interrupciones es el 8259A

Al nivel de lenguaje de máquina pueden existir instrucciones especiales para leer o escribir desde o hacia los puertos de los dispositivos de I/0 (***in***, ***out***). Esto ocurre cuando dichos puertos tienen un espacio de direcciones separado del espacio de direcciones de memoria (**I/O aisladas**). Sin embrago el caso más común es que dichos puertos estén mapeados en un espacio de direccionamiento lógico compartido con las direcciones asignadas a la memoria principal (**I/O Memory Mapped**). En este último caso el acceso a tales puertos es por medio de las instrucciones clásicas de transferencia de datos. [3]

[3] Estos tópicos se verán con más detalle en los capítulos asociados a arbitraje de bus y dispositivos de I/O.