**Pasajes de argumentos**

Métodos **para pasajes de argumentos:**

* Vía Registro: Es limitado por el número de registros. Los datos se almacenan en los registros, los cuáles deben ser documentados.
* Vía Memoria: Se guardan los datos en un área definida de la memoria (RAM).
* Vía Pila (Stack): Es independiente de la memoria y los registros.

**Pila (Stack)**

Puede ser usada por:

* El sistema: Salva la dirección de retorno cuando es llamada una subrutina o se produce una interrupción por hardware.
* El usuario: Para pasaje de parámetros. El operando está en la cabeza de la pila, la cual está apuntada por el registro SP.

**Subrutina e Interrupciones**

La **terminación** de una subrutina se produce con la instrucción RET, la cual restaura el PC; en cambio, una interrupción (por hardware) se finaliza con la instrucción IRET que, no sólo restaura el PC, sino también los flags.

La **invocación** a una subrutina se realiza a través de la instrucción CALL, a una interrupción por Software mediante la instrucción INT, y una interrupción por Hardware es llamada por un evento o señal externos, no por medio de una instrucción.

**Interrupciones**

Las interrupciones son un mecanismo mediante el cual se puede interrumpir el procesamiento normal de la CPU. Cuando un dispositivo (interno o externo) necesita la atención del procesador, envía una señal para avisarle.

Al final de cada instrucción se comprueba si se solicitó una interrupción (mediante un flag), si hay:

* Se suspende la ejecución del programa en curso.
* Guarda su contexto (próxima instrucción y estado del procesador).
* Carga el PC con la dirección de comienzo de una rutina de gestión de interrupción.
* Finaliza rutina de gestión, el procesador retoma la ejecución del programa del usuario en el punto de interrupción.

Tipos de interrupciones:

* Por Hardware: Son las verdaderas interrupciones, son generadas por dispositivos de E/S, no están relacionadas con el proceso en ejecución en ese momento.
* Traps/Excepciones: Son interrupciones por hardware creadas por el procesador en respuesta a ciertos eventos como:
  + Condiciones excepcionales (flags)
  + Falla de programa
  + Fallas de hardware
  + Accesos no alineados o zonas de memoria protegidas
* Por Software: Generalmente usadas para hacer llamados a funciones del sistemas operativo.

Interrupciones múltiples:

* Inhabilitadas 🡪 Se manejan en orden secuencial. Cuando ocurre una interrupción, se inhabilitan, se gestiona y luego se habilitan nuevamente.
* Definir prioridades 🡪 Una interrupción de mayor prioridad puede interrumpir a una de menor, se atiende y vuelve a atender a la de menor prioridad.

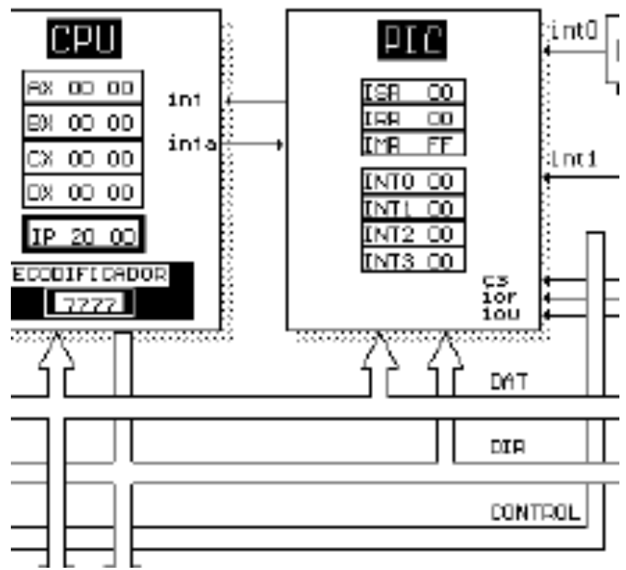
El controlador de interrupciones recibe un pedido de interrupción de un dispositivo externo y lo selecciona, luego lo envía al procesador por la línea INTR. El procesador contesta por la línea INTA y solicita un puntero de 8 bits, cuando el PIC se lo envía, éste lo utiliza para acceder a la tabla de vectores donde se encuentra la dirección del servicio, la obtiene, salva en la pila la dirección actual y los flags, y salta a la dirección del servicio.

**PIC**

El **PIC** tiene una entrada para cada dispositivo externo (int0 ... int7), por medio de estas recibe un pedido de interrupción. Envía dicho pedido al procesador por medio de la línea INTR, y recibe respuesta de éste por la línea INTA.

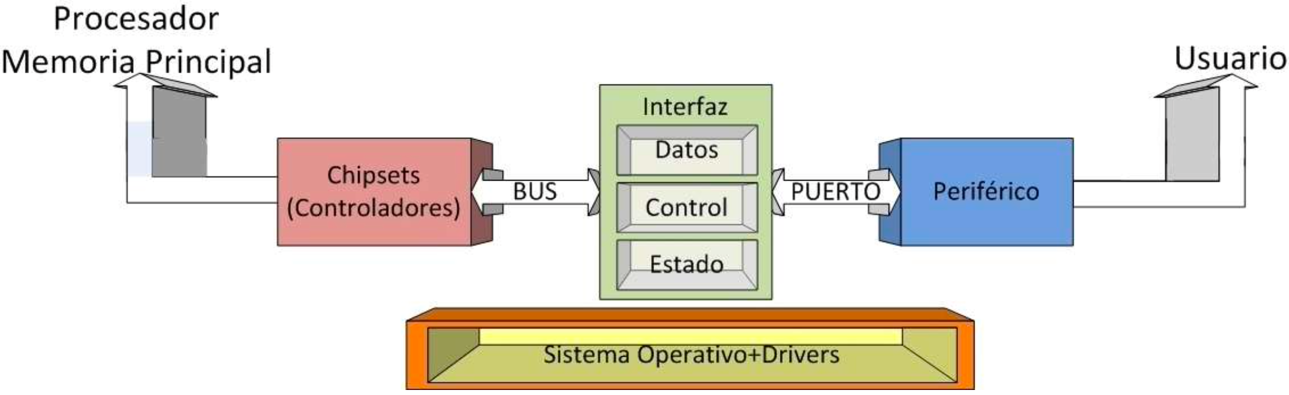
Los registros internos son (todos de 8 bits):

* INT0…INT7 🡪 Contiene el valor del vector
* ISR 🡪 Registro de interrupción de servicio
* IRR 🡪 Registro de petición de interrupción
* IMR 🡪 Registro de máscara de interrupción (si está en 1 no la atiende)
* EOI 🡪 Se envía el comando de final de interrupción



**E/S**

Un **módulo de E/S** realiza la interfaz entre el procesador y la memoria, con los periféricos.



Las técnicas de **gestión de E/S** son:

* Programada con espera de respuesta: Se produce un intercambio de datos entre la CPU y el módulo. La CPU tiene control directo sobre la operación y debe esperar a que el módulo termine, permaneciendo ociosa.
* Con interrupciones: La CPU no debe esperar a que la tarea de E/S finalice. El módulo envía un pedido de interrupción a la CPU cuando está listo.
* Acceso Directo a Memoria (DMA): La transferencia de datos se realiza entre un periférico y la memoria, sin intervención de la CPU.

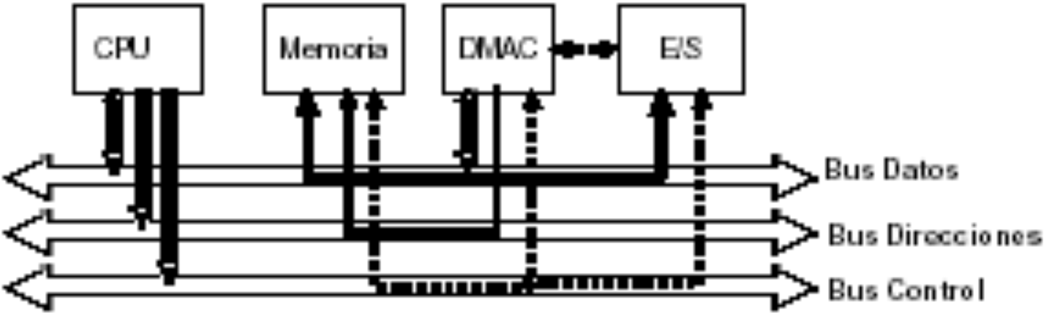
En la **gestión de E/S** programada con espera de respuesta los datos se intercambian entre la CPU y el módulo de E/S. La CPU tiene control directo sobre la operación de E/S, y debe esperar a que el módulo termine la operación, permaneciendo ociosa durante un período de tiempo.

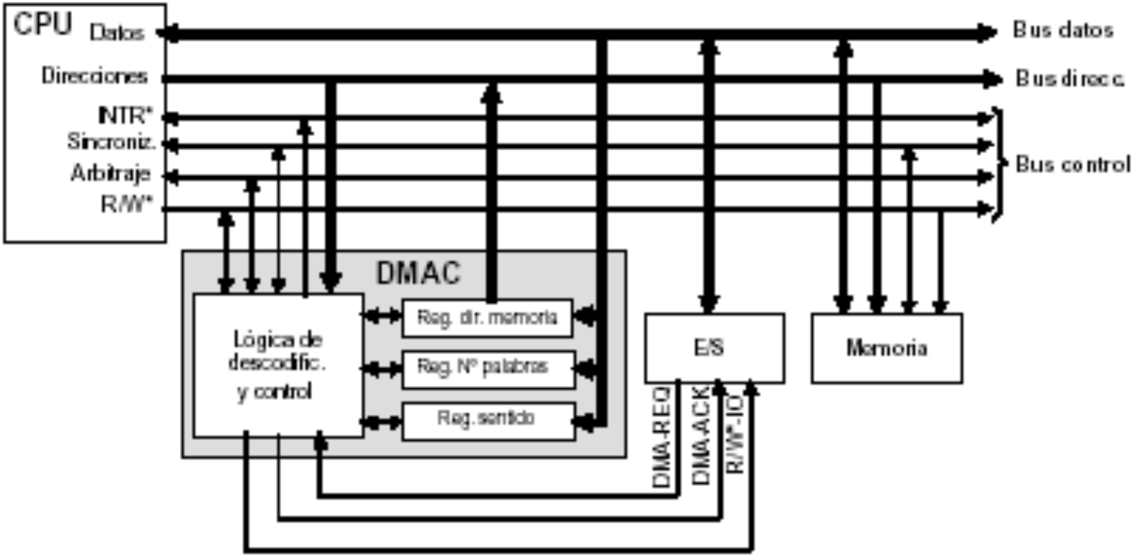
**DMA**

La coherencia de datos puede verse afectada por el uso de **DMA**, ya que si se realizan operaciones sobre un dato se actualizará en caché, y si no se vacía en la memoria antes de que un dispositivo intente accederlo, éste podría estar utilizando un valor erróneo, y viceversa.

El **DMA** (Direct Memory Access) es una técnica de gestión de E/S. El controlador de DMA (DMAC) es un dispositivo capaz de controlar una transferencia de datos entre un periférico y la memoria sin intervención de la CPU. Debe actuar como maestro del bus durante la transferencia DMA y debe:

* Solicitar el uso del bus mediante señales y la lógica de arbitraje necesarias.
* Especificar la dirección de memoria sobre la que se realiza la transferencia.
* Generar señales de control del bus
  + Tipo de operación (lectura o escritura)
  + Señal de sincronización de la transferencia





**Bus**

Un **bus** es un camino de comunicación entre dos o más dispositivos. Hay diferentes buses:

* Bus de Datos: Transmite datos e instrucciones
* Bus de Dirección: Indica la fuente o destino de un dato
* Bus de Control: Transmite información de señales de control y temporización

Tipos de buses:

* Dedicados 🡪 Uso de líneas separadas para direcciones y datos
  + 16 líneas de direcciones
  + 16 líneas de datos
  + 1 línea de control de L/E
* Multiplexados 🡪 Uso de las mismas líneas
  + 16 líneas de direcciones y datos
  + 1 línea de control de L/E
  + 1 línea de control para definir direcciones o datos

La temporización es la forma de coordinar los eventos del bus, puede ser:

* Sincrónica: La presencia de un evento está determinada por el ciclo de reloj (clock). El bus incluye una línea de reloj.
* Asincrónica: No depende del ciclo de reloj.

Los métodos de arbitraje son:

* Centralizado: Un único dispositivo hardware es responsable de asignar tiempo en el bus (controlador del bus o árbitro).
* Distribuido: Cada módulo puede controlar el acceso al bus, y dispone de la lógica para hacerlo.

**Caché**

Los elementos a tener en cuenta para el diseño de una memoria **caché** son:

* La organización de la misma (tamaño y cantidad)
* La política de ubicación (el tipo de función de correspondencia):
  + Directa 🡪 Un bloque sólo puede estar almacenado en un lugar de la caché.
  + Totalmente Asociativa 🡪 Un bloque puede almacenarse en cualquier lugar de la caché.
  + Asociativa por Conjuntos 🡪 Un bloque puede almacenarse en un conjunto restringido de lugares de la caché.
* La política de reemplazo (algoritmo de sustitución):
  + En correspondencia directa: El que ocupa el lugar del nuevo
  + En correspondencia asociativa:
    - LRU (menos recientemente usado)
    - FIFO (más antiguo)
    - LFU( menos frecuentemente usado)
    - Aleatoria: Se sustituye una línea al azar.
* Las políticas de escritura en cache dependiendo de:
* Acierto:
  + Write-through (escritura inmediata): Se actualiza simultáneamente la posición de la caché y de la memoria principal.
  + Write-back (post-escritura): La información solo se actualiza en caché. La memoria principal se actualiza en el reemplazo de datos, por lo que puede contener información errónea durante un tiempo.
* Fallo:
  + Write Allocate: La información se lleva de memoria principal a caché y se sobre-escribe en ella.
  + No-write Allocate: Se escribe directamente en memoria principal, no se lleva el bloque a caché.

**Jerarquía de memoria**

La jerarquía de memoria funciona debido a los principios de localidad referencial, los cuales cumplen los programas:

* Localidad Temporal 🡪 Los elementos de memoria referenciados recientemente volverán a serlo en un futuro próximo.
* Localidad Espacial 🡪 Los elementos de memoria cuyas direcciones están próximas a los últimos referenciados, serán referenciados en un futuro próximo.

**Segmentación de cauce**

La **segmentación de cauce** es una forma de organizar el hardware de la CPU para realizar más de una operación en paralelo. Consiste en descomponer el proceso de ejecución de las instrucciones en fases o etapas que permitan una ejecución simultánea.

Las ventajas son:

* Mejora las prestaciones a nivel de diseño hardware.
* Es invisible al programador.
* Explota el paralelismo entre las instrucciones de un flujo secuencial.
* Incrementa la productividad.

Los riesgos estructurales son:

* Cuando dos o más instrucciones intentan usar el mismo recurso de hardware a la vez, se implementa la arquitectura de Harvard, donde se utilizan memorias separadas.
* Para el problema de L/E de los registros a la vez, se implementó subdividir los ciclos, escribiendo en el primero sub-ciclo y leyendo en el segundo.

La dependencia de datos puede ser:

* Dependencia verdadera (RAW): Cuando una instrucción depende de los resultados de otra instrucción, de manera que ambas no pueden ejecutarse de forma solapada.
* Anti-dependencia (WAR): Cuando una instrucción escribe un registro y otra posterior lee ese registro. Se debe mantener el orden original para asegurarse que la última lea el valor correcto.
* Dependencia en Salida (WAW): Cuando dos instrucciones escriben en el mismo registro. Se debe mantener el orden entre las instrucciones para que el valor final sea el correcto.
* Las soluciones por software para alejar las instrucciones que generan dependencia son:
  + La inserción de instrucciones NOP
  + La reordenación de código
* Las soluciones por hardware son:
  + La detención del cauce: Espera a que el dato esté escrito en el registro para utilizarlo.
  + Forwarding (adelantamiento): Se usa el dato ni bien está disponible.

La dependencia de control se produce cuando la ejecución de una instrucción depende de cómo se ejecuta otra (**saltos**). Las soluciones pueden ser por:

* Software:
  + Implementación del salto retardado: Siempre se ejecuta la instrucción debajo del salto, y luego toma lugar el salto. Estas instrucciones son:
    - Instrucciones NOP o reordenación de código
* Hardware:
  + Detención del cauce: Hasta comprobar la condición del salto y saber dónde saltar. La penalización es de una etapa.
  + Predicción del salto: Si falla, la penalización es de un ciclo. Las técnicas son:
    - Estáticas:
      * Nunca salta: Siempre capta la siguiente instrucción
      * Siempre salta: Siempre capta la instrucción destino del salto
    - Dinámicas:
      * Conmutador saltar/no saltar: Se basa en la historia de las instrucciones. Es eficaz para los bucles.
      * Tabla de historia de saltos (Branch Target Buffer): Es una pequeña caché asociada a la etapa de búsqueda (IF)

Una instrucción de salto produce una penalización, ésta depende del tipo de salto y el tratamiento del mismo.

* Incondicional: La dirección destino se debe determinar lo más pronto posible para reducir la penalización.
* Condicional: Introduce un riesgo adicional por la dependencia entre la condición de salto y el resultado de una instrucción previa.

**Superescalares**

El paralelismo de una máquina **superescalar** depende de:

* El número de instrucciones captadas por ciclo
* El número de unidades funcionales
* Los mecanismos para localizar instrucciones independientes:
  + Identificar paralelismo y organizar F, D y E en paralelo
  + Renombre de registros
  + Ventana de instrucciones

El objetivo de usar la técnica de **renombre de registros** es evitar la serialización innecesaria de las operaciones de los programas impuesta por la reutilización de los registros del procesador.

Características de un procesador superescalar:

* Se pueden llevar a cabo más de una instrucción en simultáneo.
* Conlleva la duplicación de algunas o todas las partes del CPU/ALU.
* El grado de paralelismo y la aceleración de la máquina aumentan.

Políticas de emisión:

* Emisión y finalización en orden
* Emisión en orden y finalización desordenada
* Emisión y finalización desordenadas

**MIMD**

**MIMD** (Multiple Instruction Multiple Data) es un conjunto de procesadores que ejecutan secuencias de instrucciones diferentes en simultáneo, con un conjunto de datos diferentes. Se pueden dividir según la forma de comunicarse:

* Memoria compartida:
  + SMP
  + Sistemas NUMA
* Memoria distribuida:
  + Clusters

**Clusters**

Las características de un Cluster:

* Son computadoras completas interconectadas que trabajan conjuntamente como un único recurso.
* Cada computadora se denomina “nodo”.
* Prestaciones y disponibilidades elevadas.
* Aplicaciones propias de un servidor.

**SMP**

Las características de un multiprocesador simétrico (SMP) son:

* Dos o más procesadores similares de capacidades comparables.
* Comparten la memoria principal y las E/S.
* Interconectados mediante un bus u otro tipo de sistema de interconexión.
* Tiempo de acceso a memoria similar para todos (UMA).
* Todos los procesadores pueden desempeñar las mismas funciones.
* Sistema operativo integrado, que proporciona la interacción entre los procesadores y sus programas.

**SMP y clusters**

Las características que **diferencian** los **SMP** de los **clusters** son:

* Son más fácil de administrar y configurar.
* Son cercanos a los sistemas de un solo procesador.
* Ocupan menos espacio físico, y por lo tanto tienen menor consumo.
* Los SMP comparten la memoria principal y la E/S, mientras que los clusters no comparten nada, sólo se conectan mediante un bus o red de interconexión.

**RISC y CISC**

No existe un par de máquinas RISC y CISC directamente comparables, la mayoría de las comparaciones se realizan con máquinas de “juguete”. La mayoría de las máquinas son una mezcla de ambas.

|  |  |
| --- | --- |
| **RISC** | **CISC** |
| Pocas instrucciones simple | Muchas instrucciones complejas |
| Instrucciones de longitud fija | Instrucciones de longitud variable |
| Complejidad en el compilador | Complejidad en el microprograma |
| Acceso a memoria solo con load/store | Acceso a memoria mediante muchas instrucciones |
| Pocos modos de direccionamiento | Muchos modos de direccionamiento |