**Documento 1: Arquitectura de Computadoras - Esquema Ordenador**

**1. Conceptos de Computadora**

* **Computadora**: Es una máquina electrónica que realiza tareas específicas. Estas tareas incluyen:
  + Aceptar información (entrada de datos).
  + Almacenar información (en memoria).
  + Procesar información siguiendo un conjunto de instrucciones (ejecutar un programa).
  + Producir resultados (salida de datos).

**2. Organización y Arquitectura**

* **Organización**: Describe cómo se implementa físicamente el diseño de una computadora. Esto incluye:
  + **Interconexión de componentes**: Cómo se conectan las unidades de procesamiento, memoria y dispositivos de entrada/salida (E/S).
  + **Unidad de control**: Cómo se controla el flujo de datos entre los componentes.
  + **Acceso y gestión de memoria**: Cómo se accede a la memoria y se organizan los datos.
* **Arquitectura**: Se refiere al diseño lógico y conceptual de un sistema informático. Aquí se incluye:
  + **Conjunto de instrucciones**: Las instrucciones que puede ejecutar el procesador.
  + **Organización de la memoria**: Cómo se organizan y acceden los datos en la memoria.
  + **Unidades de procesamiento**: Cómo se manejan las operaciones aritméticas y lógicas.

**3. Diferencias entre Organización y Arquitectura**

* **Ejemplo**: Los procesadores **Intel64** y **AMD64** tienen una **organización** diferente (es decir, la forma en que se conectan y operan sus componentes), pero ambos comparten una arquitectura común llamada **x86-64** (es decir, el diseño lógico de las instrucciones y la memoria).

**4. Tipos de Arquitecturas**

**4.1. Arquitectura Von Neumann**

* En esta arquitectura, hay un único espacio de memoria que contiene tanto las instrucciones como los datos.
* Las **propiedades** clave de la arquitectura Von Neumann son:
  + **Espacio único de memoria** para datos e instrucciones.
  + **Acceso secuencial**: La ejecución de las instrucciones es secuencial.
  + **Direcciones de memoria** accesibles por su posición, independientemente de si se accede a datos o a instrucciones.

**4.2. Arquitectura Harvard**

* A diferencia de Von Neumann, **divide la memoria** en dos partes separadas: una para datos y otra para instrucciones.
* **Accesos simultáneos**: El procesador puede acceder tanto a la memoria de datos como a la de instrucciones al mismo tiempo.
* Esta arquitectura se utiliza más en sistemas embebidos (como microcontroladores) y no tanto en computadoras de propósito general.

**5. Ley de Moore**

* La **Ley de Moore** fue propuesta por Gordon Moore (cofundador de Intel) en 1965. Dice que:
  + La cantidad de **transistores** en un chip se **duplica aproximadamente cada dos años**.
  + Esta observación ha llevado al crecimiento exponencial de la capacidad de procesamiento de las computadoras.

**5.1. Implicaciones de la Ley de Moore:**

* **Mayor potencia de procesamiento**: El aumento en la cantidad de transistores hace que los procesadores sean más potentes.
* **Miniaturización**: A medida que los transistores se vuelven más pequeños, se puede reducir el tamaño de los dispositivos electrónicos.
* **Reducción de costos**: El costo por transistor ha disminuido, lo que hace que la tecnología sea más asequible.

**5.2. Evolución de los transistores:**

* **Intel 4004 (1971)**: Tamaño de transistor de 10 micrómetros.
* **Década de los 2000**: Procesadores como **Intel Pentium 4** y **AMD Athlon XP** con transistores de 0.18 micrómetros.
* **Década de los 2010**: Procesadores avanzados con transistores de 45 nm (como Intel Core) y posteriormente de 32 nm.
* **Década de los 2020**: Procesadores como el **AMD Ryzen 5000** y **Apple M1** tienen transistores de 7 nm.

**6. Esquema Ordenador**

**6.1. Buses**

* **Buses**: Son conjuntos de cables que **transportan información binaria** entre diferentes componentes de la computadora. Existen tres tipos de buses:
  1. **Bus de datos**: Transfiere los datos entre la CPU, la memoria y los dispositivos.
  2. **Bus de direcciones**: Transporta la dirección de la posición de memoria o periférico a donde se enviarán o recibirán los datos.
  3. **Bus de control**: Controla la temporización de eventos durante las transferencias y sincroniza la comunicación entre los componentes.
* **Ejemplo**: Si la CPU necesita escribir en la memoria, usa el bus de control para enviar la señal que indica la operación de escritura, el bus de direcciones para indicar la ubicación de la memoria y el bus de datos para enviar el valor.

**6.2. Reloj del Esquema Ordenador**

* **Reloj**: Marca la sincronización de trabajo de los componentes internos de la computadora.
* La **unidad de medida** de la frecuencia del reloj es el **Hertz** (Hz), y la mayoría de los componentes internos de la computadora trabajan sincronizados con este reloj.

**6.3. Ciclo de Instrucción**

* El ciclo básico para ejecutar una instrucción es:
  1. **Leer**: Se lee la instrucción desde la memoria.
  2. **Interpretar**: La CPU decodifica la instrucción para entender qué operación realizar.
  3. **Ejecutar**: Se ejecuta la operación indicada por la instrucción.
* **ALU (Unidad Aritmético-Lógica)**: Realiza las operaciones matemáticas y lógicas.
* **Unidad de Control**: Coordina la ejecución de las instrucciones y genera las señales de control necesarias.

**7. Instrucciones del Procesador**

* **Tipos de Instrucciones**:
  + **Instrucciones de movimiento de bits**: Manipulan el orden de los bits dentro de un dato.
  + **Instrucciones aritméticas**: Realizan operaciones matemáticas y lógicas.
  + **Instrucciones de salto**: Cambian el valor del contador de programa, permitiendo la ejecución no secuencial de código.
  + **Instrucciones a memoria**: Permiten que el procesador lea y escriba datos en la memoria.

**8. Unidades de Control: Cableada vs Microprogramada**

**8.1. Unidad de Control Cableada**

* Utiliza **circuitos lógicos fijos** (hardware) para decodificar las instrucciones y generar señales de control.
* **Ventajas**: Es más rápida porque las señales de control se generan directamente por el hardware.
* **Desventajas**: Menos flexible, ya que cambiar el comportamiento del sistema requiere modificar físicamente el hardware.

**8.2. Unidad de Control Microprogramada**

* Utiliza un enfoque más flexible, donde las señales de control se generan mediante **microinstrucciones** almacenadas en una memoria de control.
* **Ventajas**: Más flexible, ya que se puede cambiar el comportamiento del procesador actualizando las microinstrucciones sin necesidad de modificar el hardware.
* **Desventajas**: Más lenta que la unidad cableada, ya que las señales de control se leen desde la memoria de control.

**Documento 2: El Microprocesador 8086**

### ****1. Introducción al 8086****

* El **8086** es un microprocesador **de 16 bits** que fue desarrollado por Intel.
* Usa un **bus de direcciones de 20 bits**, lo que le permite acceder hasta **1 MB de memoria**.
* Está basado en la **arquitectura Von Neumann**, lo que significa que los datos y las instrucciones se almacenan en el mismo espacio de memoria.

### ****2. Características del 8086****

* **Procesador de 16 bits**: El 8086 procesa datos en palabras de 16 bits.
* **Bus de datos de 16 bits**: Esto significa que puede transferir 16 bits de datos por ciclo.
* **Bus de direcciones de 20 bits**: Le permite acceder a hasta 1 MB de memoria.
* **Registros de propósito general**:
  + El 8086 tiene **8 registros de propósito general** (AX, BX, CX, DX, SI, DI, BP, SP), cada uno de **16 bits**.
  + Estos registros se pueden dividir en dos partes de 8 bits (por ejemplo, AX se divide en AH y AL).

### ****3. Registros del 8086****

#### 3.1. Registros de Propósito General

* **AX (Acumulador)**:
  + Usado para almacenar resultados de operaciones aritméticas y lógicas.
  + Es el único registro que puede ser usado para multiplicaciones y divisiones.
  + Se puede dividir en **AH** (parte alta) y **AL** (parte baja).
* **BX (Base)**:
  + Usado como registro base en operaciones de acceso a memoria.
  + Puede accederse como **BH** (parte alta) y **BL** (parte baja).
* **CX (Contador)**:
  + Actúa como contador en bucles y operaciones de desplazamiento.
  + **CL** se usa para operaciones de desplazamiento y rotación de bits.
* **DX (Datos)**:
  + Se utiliza para almacenar datos involucrados en operaciones como entrada/salida o multiplicaciones.

#### 3.2. Registros de Segmento

* Los registros de segmento contienen valores que apuntan a segmentos de memoria. Cada uno tiene una función específica:
  + **CS (Segmento de código)**: Contiene la dirección del segmento que almacena el código del programa. Se usa junto con el registro **IP** para obtener la dirección de la siguiente instrucción.
  + **DS (Segmento de datos)**: Contiene la dirección del segmento donde se almacenan los datos.
  + **ES (Segmento extra)**: Un registro extra que puede ser usado para acceder a otro segmento de datos.
  + **SS (Segmento de pila)**: Contiene la dirección del segmento donde está la pila. Trabaja junto con el registro **SP**.

#### 3.3. Registros de Índice

* **IP (Puntero de Instrucción)**: Contiene el desplazamiento dentro del segmento de código y apunta a la siguiente instrucción que será ejecutada.
* **SI (Índice de Origen)**: Usado como desplazamiento para acceder a datos en operaciones de cadena (string operations).
* **DI (Índice de Destino)**: Usado como desplazamiento para escribir datos en operaciones de cadena.
* **SP (Puntero de Pila)**: Usado junto con **SS** para acceder a la pila. Contiene el desplazamiento al último valor almacenado en la pila.
* **BP (Puntero Base)**: Utilizado como un puntero adicional a la pila o a otros segmentos de datos. Por defecto, opera en el segmento de pila.

#### 3.4. Registro de Estado

* El **registro de estado** contiene varias banderas que indican el estado del procesador:
  + **Carry flag (CF)**: Indica un acarreo en operaciones aritméticas.
  + **Zero flag (ZF)**: Se activa cuando el resultado de una operación es cero.
  + **Sign flag (SF)**: Indica si el resultado de una operación es positivo o negativo.
  + **Overflow flag (OF)**: Indica desbordamiento en operaciones aritméticas con signo.
  + **Interrupt flag (IF)**: Permite o bloquea las interrupciones.

### ****4. Modos de Direccionamiento del 8086****

Los modos de direccionamiento son las diferentes maneras en que una instrucción puede especificar dónde encontrar los operandos (datos).

#### 4.1. Modos de Direccionamiento Básicos

* **Implícito**: El operando está implícito en la propia instrucción.
  + Ejemplo: STI (Set Interrupts) no especifica operando.
* **Inmediato**: El operando es una constante dentro de la instrucción.
  + Ejemplo: MOV AX, 500 carga el valor **500** en el registro **AX**.
* **Registro**: El operando es un registro.
  + Ejemplo: MOV AX, BX transfiere el contenido de **BX** a **AX**.
* **Directo**: El operando está en una dirección de memoria.
  + Ejemplo: MOV BX, [1000] almacena en **BX** el contenido de la dirección de memoria **1000**.

#### 4.2. Modos de Direccionamiento Indirectos

* **Base**: Usa un registro base para apuntar a una dirección en memoria.
  + Ejemplo: MOV AX, [BX] almacena en **AX** el contenido de la dirección apuntada por **BX**.
* **Índice**: Usa un registro de índice para apuntar a una dirección en memoria.
  + Ejemplo: MOV AX, [SI] almacena en **AX** el contenido de la dirección apuntada por **SI**.
* **Base + Desplazamiento**: Usa un registro base y un desplazamiento para calcular la dirección.
  + Ejemplo: MOV AX, [BP + 7] almacena en **AX** el contenido de la dirección apuntada por **BP** más **7**.
* **Índice + Desplazamiento**: Usa un registro de índice y un desplazamiento.
  + Ejemplo: MOV AX, [DI + 7] almacena en **AX** el contenido de la dirección apuntada por **DI** más **7**.
* **Base + Índice**: Usa un registro base y un registro de índice para calcular la dirección.
  + Ejemplo: MOV AX, [BX + SI] almacena en **AX** el contenido de la dirección resultante de sumar **BX** y **SI**.
* **Base + Índice + Desplazamiento**: Combina un registro base, un índice y un desplazamiento.
  + Ejemplo: MOV AX, [BX + SI + 9] almacena en **AX** el contenido de la dirección resultante de sumar **BX**, **SI** y **9**.

### ****5. Segmentación de Memoria en el 8086****

* El 8086 tiene un bus de direcciones de **20 bits**, lo que le permite acceder a hasta **1 MB** de memoria (2^20 = 1,048,576 bytes).
* Los registros internos del procesador son de **16 bits**, por lo que no pueden manejar directamente direcciones de 20 bits.
* Para resolver esto, se utiliza la **segmentación**:
  + La memoria se divide en segmentos de **64 KB**.
  + Cada dirección de memoria se especifica como un **segmento** y un **desplazamiento** dentro de ese segmento.
  + La dirección física de la memoria se calcula combinando el valor del segmento y el desplazamiento.

### ****6. Pipelining en el 8086****

* **Pipelining**: Es una técnica de procesamiento en paralelo que permite ejecutar varias instrucciones al mismo tiempo, solapando etapas de su ejecución.
* En el 8086, el pipelining es posible gracias a la cola de instrucciones que permite a la **Unidad de Ejecución (UE)** y la **Unidad de Interfaz de Bus (UIB)** trabajar de manera simultánea.
  + La **UIB** obtiene las siguientes instrucciones mientras la **UE** ejecuta la instrucción actual.
* **Ventajas del pipelining**:
  + Mejora el rendimiento porque reduce los tiempos de espera de la **UE**.
  + Acelera el procesamiento al llenar la cola de instrucciones para que siempre haya una instrucción lista para ejecutarse.

#### 6.1. Hazards (Peligros) en el Pipeline

* **Hazards estructurales**: Ocurren cuando dos instrucciones intentan usar el mismo recurso de hardware al mismo tiempo.
  + Soluciones: Generar una señal de "stall" (espera) o agregar más hardware.
* **Hazards de datos**: Ocurren cuando una instrucción depende de los resultados de una instrucción anterior que aún no ha terminado.
  + Tipos de hazards de datos:
    - **Read after Write (RaW)**: Una instrucción intenta leer un dato que aún no ha sido escrito.
    - **Write after Read (WaR)**: Una instrucción escribe un dato antes de que una instrucción anterior lo haya leído.
    - **Write after Write (WaW)**: Dos instrucciones escriben en el mismo operando, y la primera termina después que la segunda.
* **Hazards de control**: Ocurren cuando una instrucción de salto altera el flujo del programa, impidiendo que el pipeline continúe ejecutando las instrucciones en el orden previsto.

**Documento 3: Interrupciones**

**1. Introducción a las Interrupciones**

* Una **interrupción** es una señal enviada al procesador que indica que debe detener temporalmente la ejecución de la instrucción actual para pasar a ejecutar una tarea especial, llamada **rutina de servicio de interrupción** (ISR, por sus siglas en inglés).
* **Ventajas** de usar interrupciones:
  + Permiten que el procesador responda a eventos externos (como pulsaciones de teclado o datos recibidos por un puerto) de manera eficiente.
  + Son más eficientes que el **polling** (sondeo), ya que el procesador no tiene que verificar constantemente si hay eventos pendientes; en lugar de eso, responde solo cuando ocurre una interrupción.

**2. Polling vs. Interrupciones**

* **Polling**: Es una técnica en la que el procesador verifica repetidamente si un dispositivo periférico necesita atención. Esta técnica es ineficiente porque consume mucho tiempo de procesamiento.
* **Interrupciones**: El dispositivo periférico genera una señal de interrupción para notificar al procesador solo cuando requiere atención. Esto permite utilizar mejor el tiempo del procesador.

**3. Tipos de Interrupciones**

Las interrupciones se pueden clasificar según su origen:

**3.1. Interrupciones por Hardware**

* **Internas**: Generadas dentro de la CPU por eventos como:
  + **División por cero**.
  + **Desbordamiento** (overflow).
  + **Instrucción ilegal** (código no válido).
* **Externas**: Generadas por dispositivos de entrada/salida (I/O) y pueden ser:
  + **Vectorizadas**: Asociadas con una dirección fija en la memoria donde se encuentra la rutina de servicio.
  + **No vectorizadas**: No tienen una dirección fija, por lo que el procesador debe buscar la dirección de la rutina de servicio.

**3.2. Interrupciones por Software**

* Generadas por el propio código del programa, por ejemplo, cuando se ejecuta una instrucción **INT** que provoca una interrupción.

**4. Tipos de Interrupciones por Hardware**

* **Interrupciones enmascarables**: Estas interrupciones pueden ser habilitadas o deshabilitadas por el usuario mediante el uso de una bandera especial.
* **Interrupciones no enmascarables (NMI)**: Estas interrupciones tienen prioridad sobre las enmascarables y no pueden ser desactivadas. Son utilizadas en situaciones críticas, como en casos de fallos graves del sistema.

**5. Respuesta del Procesador a una Interrupción en el 8086**

Cuando ocurre una interrupción, el 8086 sigue una secuencia específica para manejarla:

**5.1. Etapas de la Respuesta a una Interrupción**

1. **Guardado del estado actual**:
   * El procesador empila los registros importantes: **CS** (Segmento de Código), **IP** (Puntero de Instrucción) y el **registro de estado** en la pila. Esto se hace para poder restaurar el estado original cuando termine de atender la interrupción.
2. **Carga de la dirección del manejador de interrupción**:
   * El procesador carga una nueva dirección en **CS** e **IP** para apuntar a la **rutina de servicio de interrupción** (ISR) correspondiente. Esta dirección se encuentra en una tabla especial llamada **vector de interrupciones**.
3. **Ejecución de la rutina de interrupción**:
   * El procesador ejecuta el código de la ISR, que se encarga de manejar la interrupción.
4. **Retorno de la interrupción**:
   * Una vez que la rutina ha terminado, se ejecuta la instrucción **IRET** (Return from Interrupt), que desempila el registro de estado, **IP**, y **CS** para restaurar el estado original y continuar la ejecución del programa interrumpido.

**6. Vector de Interrupciones**

* El **vector de interrupciones** es una tabla que se encuentra en las primeras 1024 posiciones de la memoria (direcciones **00000H** a **003FFH**). Cada interrupción tiene un **número** o **tipo** asociado que apunta a una dirección en la memoria donde se encuentra la rutina de servicio.
* Hay un total de **256 posibles interrupciones** en el 8086, y cada interrupción tiene un **puntero** de 4 bytes (16 bits para el segmento y 16 bits para el desplazamiento). Este puntero indica la dirección donde comienza la rutina de servicio para esa interrupción.

**Ejemplo de Vectorización:**

* Si ocurre la interrupción **número 4**, el procesador buscará la dirección de la rutina de servicio en la posición **4** de la tabla de interrupciones.

**7. Controlador de Interrupciones 8259**

* El **controlador de interrupciones programable (PIC) 8259** es un dispositivo que se encarga de gestionar múltiples solicitudes de interrupción y enviarlas al procesador.

**7.1. Funcionamiento del 8259:**

* Un dispositivo genera una señal de interrupción.
* El **8259** verifica si no hay una interrupción de mayor prioridad en ejecución o pendiente.
* Si la interrupción puede ser atendida, el **8259** envía la señal de interrupción **INTR** al procesador.
* Si el **8086** acepta la interrupción, responde enviando una señal de aceptación **INTA** al **8259**.
* El **8259** coloca en el bus de datos el **identificador del dispositivo** que solicitó la interrupción.

**7.2. Prioridades en el Controlador 8259:**

* El **8259** puede manejar hasta **8 niveles de prioridad**.
* El controlador decide qué interrupción tiene mayor prioridad y debe ser atendida primero.

**8. Interrupciones en el 8086: Ejemplo Completo**

Supongamos que un dispositivo externo genera una señal de interrupción mientras el procesador está ejecutando un programa:

1. El dispositivo envía una señal de **INTR** al procesador.
2. Si no hay interrupciones de mayor prioridad, el **8259** acepta la interrupción y notifica al **8086**.
3. El **8086** detiene la ejecución del programa y guarda los registros **CS**, **IP** y el registro de estado en la pila.
4. El **8086** busca en el vector de interrupciones la dirección de la rutina de servicio correspondiente.
5. Se ejecuta la rutina de servicio.
6. Después de que la rutina termina, el **8086** ejecuta la instrucción **IRET**, lo que restaura los registros **CS**, **IP** y el registro de estado, y luego continúa ejecutando el programa original.