# Arquitectura de Computadores

**Tema N° 1: Introducción a la Arq. de Computadoras**

Eduardo Daniel Cohen – dcohen@arnet.com.ar

http://www.herrera.unt.edu.ar/arqcom

# Temas a Tratar: Introducción

- Información General del Curso
- Componentes de un Computador.
  - I/O, Memoria, CPU, Buses.
  - Jerarquía de Memoria.
  - Sistema de Buses.
  - CPU – Camino de Datos - Control
  - El ciclo de la Instrucción
  - Todo junto: ejecución de un programa.
- Conceptos de Sw
  - Lenguaje de Máquina, Lenguaje Superior.
  - Sistema Operativo, Ensamblador, Intérprete, Compilador
- Interrupciones y Excepciones.

# ¿Qué estaremos viendo?

- **Módulo I**
  - **Introducción. ¿qué es una computadora?**
    - **CPU – Memoria – I/O.**
    - **Modelo Jerárquico "por capas"**
  - **Lenguaje Ensamblador – MIPS R3000**
  - **Performance de Computadoras.**
  - **Paralelismo y Trabajo en Serie.**
  - **Ensamblador, linker, loader.**
  - **Interrupciones y Excepciones.**

- **Módulo II**
  - **Diseño del Procesador.**
    - **Procesador de ciclo único.**
    - **Procesador Segmentado.**
    - **Procesadores Superescalares.**
  - **Sistema de Memoria.**
    - **Caché.**
    - **Memoria Virtual.**
  - **Buses.**
  - **Sistema de I/O.**
    - **Dispositivos**

# Bibliografía

**Libro de Texto Principal:**

Patterson, D. A.; Hennessy, J. L.
Estructura y diseño de Computadores: Interficie circuitería/programación. Vol. 1 y 2. Reverté, 2000


**Principales Transparencias:**

Adaptadas de clases del curso CS152, Universidad de Berkeley, California.
Prof. D. A. Patterson

**Cuadernillo de Problemas de la Cátedra**

**Recomendación: leer antes de clase.**

# Objetivos

- **Temas a Aprender**
  - **Cómo funcionan las computadoras, fundamentos.**
  - **Como analizar su performance (¡o como no hacerlo!)**
  - **Temas relacionados con procesadores modernos (caches, pipelines).**
  - **Criterios de Diseño.**
  - **Ritmo y Actualización → Tomado del Curso en Berkeley 2001.**
- **¿Para qué aprender estos temas?**
  - **Imprescindible para "Ingenieros en Computación"**
  - **Globalización: se podrá trabajar en Argentina para Cías en USA.**
  - **Deben manejar la "performance" de las instalaciones: Hw y Sw de Base.**
  - **Necesitarán tomar decisiones de compra de equipos y ofrecer asesoramiento profesional.**

# Funcionamiento del Curso - I

- **Dictado Conceptual – Ing. Daniel Cohen**
  - **3 clases por semana en general – de 2 hs c/u. (L, M, V de 8 a 10 hs).**
  - **Importante: Venir a Clase.**
  - **Consultas después de cada clase.**
  - **Ing. Daniel Cohen - dcohen@arnet.com.ar**

# Funcionamiento del Curso - II

- **Trabajos Prácticos.**
  - **Ing. Nicolás Majorel Padilla npadilla@herrera.unt.edu.ar**
  - **Ing. Esteban Volentini – evolentini@herrera.unt.edu.ar**
  - **Juan Esteban Maldonado – jemaldonado80@yahoo.com.ar**
  - **Raúl Criado – raulo@cqc.tv**
  - **Ramiro Orso – orsoram@uol.com.ar**
  - **Pablo Mazzeo – pablomazzeo@hotmail.com**
  - **1 vez por semana – Problemas a resolver individualmente.**
  - **Evaluación de cada práctico – 1 vez por semana.**
  - **Horarios: Lunes y Viernes de 10 a 11 hs.**
  - **Consultas a Estudiantes 2 hs dos veces por semana.**

# Funcionamiento del Curso III

- **3 Trabajos de Laboratorio.**
  - **Trabajo en Grupos RTN.**
  - **Uso del Simulador SPIM - Trabajo en assembler del MIPS.**
  - **Uso del Simulador de Memoria Cache (CACHESIM)**
- **Regularidad**
  - **Aprobación Evaluaciones TP y Laboratorios prom.>40%**
  - **Aprobar todos los trabajos de Laboratorio**
  - **Asistencia 80% evaluaciones (solo una se puede faltar)**
  - **Parciales con promedio > 40%.**
  - **Aprobar el parcial N° 3 y al menos dos parciales.**

# Funcionamiento del Curso IV

- **Evaluación Distribuida.**
  - **3 Parciales (60%).**
  - **Evaluación Prácticos y Laboratorios (30%)**
  - **Participación en clase, Quizzes (10%)**
  - **Porcentajes tentativos, cambian si aparecen “trampas”.**
  - **Quien copia recibe “0”.**

# Funcionamiento del Curso V

- **Correlativas a aprobar para Inscribirse:**
  - Probabilidad y Estadísticas.
  - Estructura de Datos I.
  - Electrónica II.
- **Arquitectura es obligatoria para:**
  - Sistemas Operativos (y muchas que le siguen).
  - Diseño con Microprocesadores.
  - Laboratorio de Microprocesadores

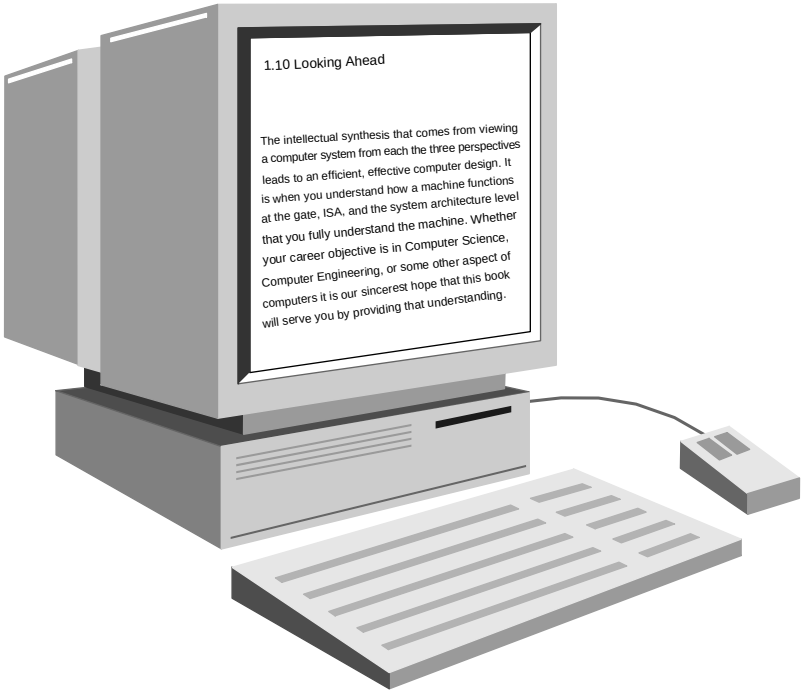
# Página de la Cátedra

- **Material de Ayuda – transparencias**
- **Anuncios y Novedades.**
- **Información General – Programa de la Materia.**
- **Trabajos Prácticos de Ejercitación y Soluciones (a la semana).**
- **Enunciados de Evaluativos y sus Soluciones, años anteriores**
- **Evaluativos y Soluciones 2005 – (a la semana)**
- **Grupo de Discusión – preguntas y respuestas para todos.**
- **Chat.**
- **Encuestas y quejas anónimas... La opinión de los estudiantes nos ayuda.**
- **Calificaciones.**
- **INSCRIBIRSE EN LA PAGINA.**
- **Página de Argentina más accedida por Hispano-parlantes.**

# Novedad 2005: Tutorías

- Una materia compleja y muy diferente.
- Tratamos de mejorar posibilidades de éxito de los estudiantes
- **2005 – Tutorías.**
  - Para ayudar en el cómo y no en el qué.
  - El Tutor no toma decisiones ni resuelve – ayuda a que el discípulo pueda hacerlo.
  - Se trata de Contención, no Contenidos.
  - Compromiso de una reunión semanal corta de seguimiento.
- **Reglas:**
  - 2 Grupos para comparar: uno con y otro sin.
  - Los que estén sin tutoría quedan a mi cargo y yo estoy a disposición para ayudarles.
  - Las tutorías no tienen ninguna influencia en la nota.

# Componentes de un Computador




**Desde Afuera:**

- **Periféricos**
  - **Mouse**
  - **Teclado**
  - **Monitor**
  - **Discos – Disketes**
  - **Otros...**
- **Input / Output**
  - **¿Cuál es Input?**
  - **¿Output?**
  - **¿Input y Output?**
  - **¿Variedad de velocidades?**
  - **Comunica con el Mundo.**

# Por dentro...

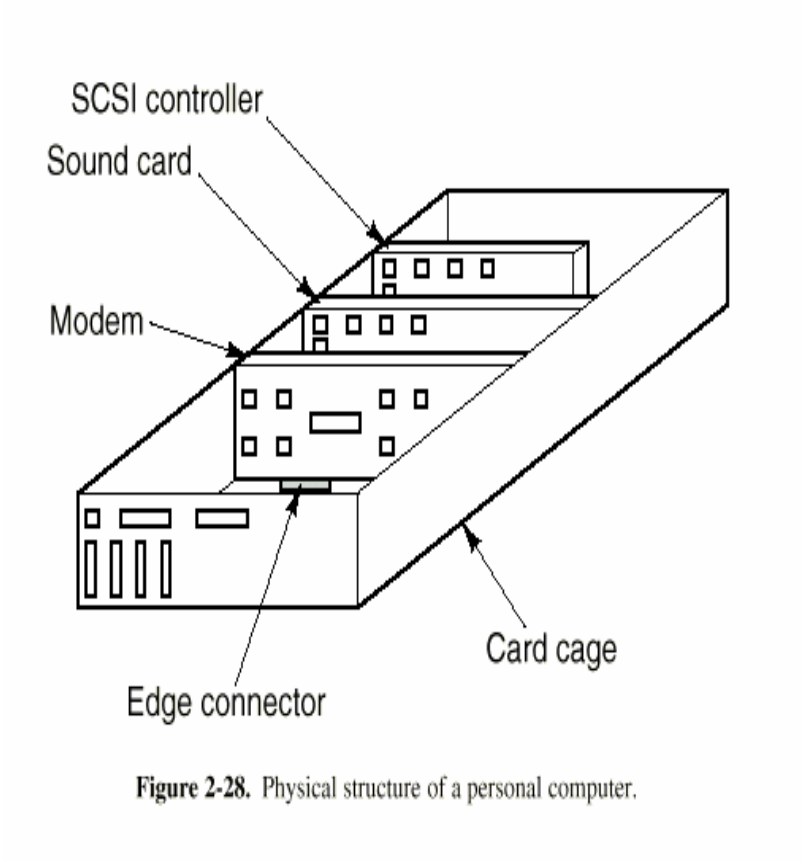
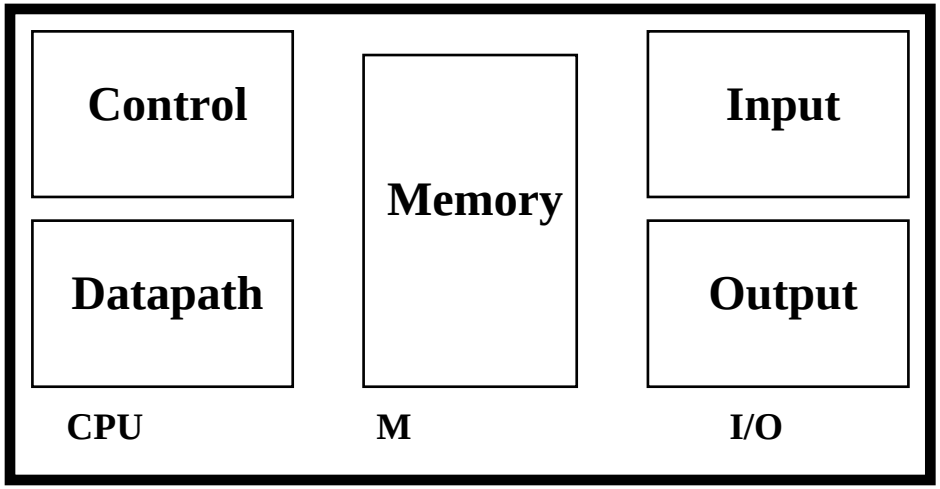



**Figure 2-28.** Physical structure of a personal computer.

- **Motherboard**
  - **Procesador (CPU)**
  - **Memoria (M)**
  - **Conectores - Bus**
  - **Plaquetas de Expansión (I/O)**
    - **Modem.**
    - **Placa Multimedia.**
    - **Otros...**

# Partes de un Computador

- **CPU – sistema complejo.**
- **Dividir Funcionalmente: Acción (camino de Datos) y Control.**
- **En síntesis:**



# Memoria - M

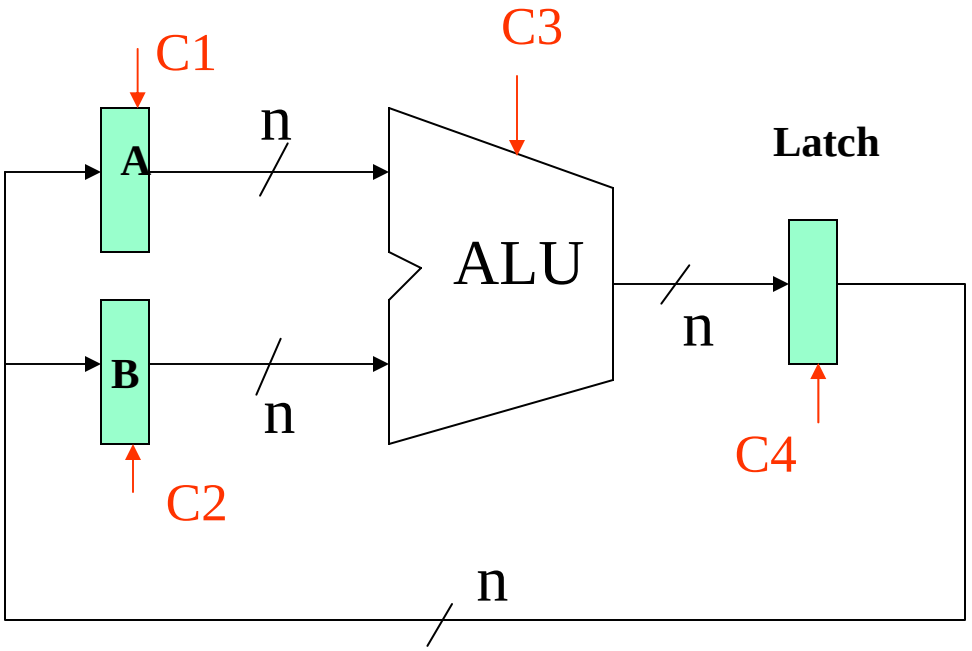
- **Contiene DATOS y PROGRAMAS almacenados.**
- **Ejemplo de Datos (X e Y)**
  - **Posición A – X**
  - **Posición B – Y**
- **Ejemplo de Programa:**
  - **Tome X de A**
  - **Súmele Y de B**
  - **Guarde el resultado en C.**
- **Concepto de Programa Almacenado**
  - **Puede ser manipulado como datos por otro programa.**

# CPU - Procesador

- **Ejecuta los Programas.**
  - **Lee Instrucciones de Memoria**
  - **Las Decodifica**
  - **Las Ejecuta**
    - **Lee / Escribe Datos de Memoria**
    - **Recibe / Envía Datos a I/O**
    - **Realiza Operaciones Lógico-Artiméticas (PROCESA)**
- **Se compone de**
  - **Unidad de Control – fija qué se debe hacer y cuándo.**
  - **Camino de Datos: Registros, ALU, interconexión.**
    - **Se encarga de hacer.**
  - **Analogía: Sistema Nervioso y Músculos.**

# Camino de Datos

- **Ejemplo Simple**



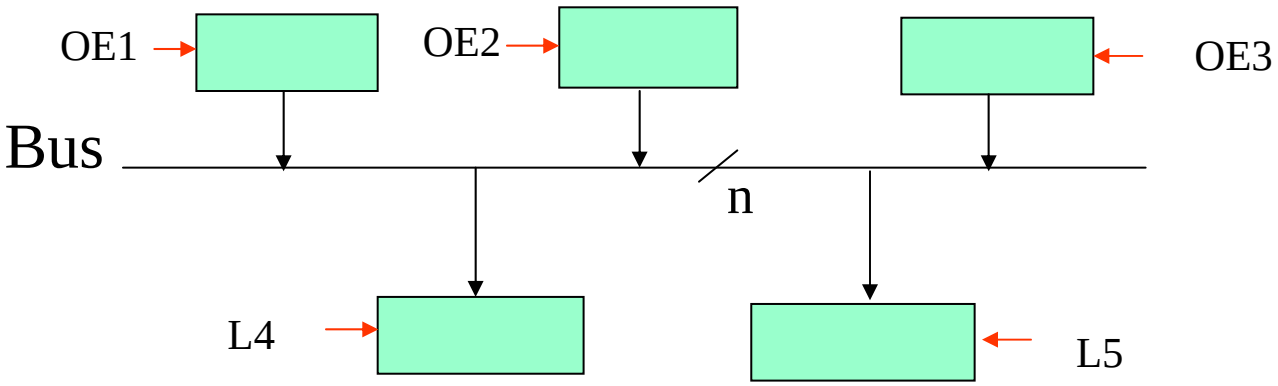

- **C1, C2, C3 y C4 son señales de control.**
- **¿De qué depende la cantidad de bits de C3?**
- **¿Para qué está el LATCH?**

# Camino de Datos – Ejemplo más elaborado

- **Se desea interconectar 7 Registros y un ALU.**
  - **R1, R2, ..., R7 y ALU.**
  - **Intercambiar Contenidos de Registros (Ri <-> Rj)**
  - **Acceder al ALU desde cualquier Registro (Ri→ALU)**
  - **Poner el resultado en cualquier Registro (ALU → Rj)**
- **Solución 1**
  - **Conectar todos contra todos.**
  - **Evaluar Ventajas y Desventajas.**

# Concepto de Bus

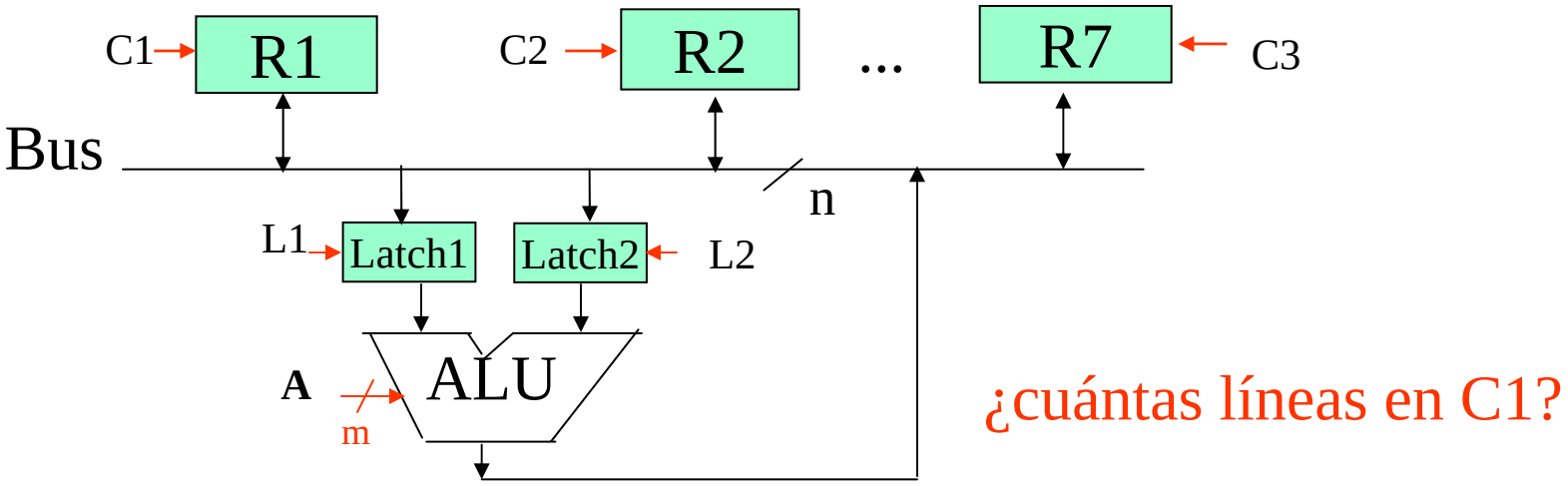
- **Salida 3 State:**
  - **Posibilita conexión de salidas en forma directa a un conjunto de conectores = bus.**
  - **Se sube con OE=1 (output-enable)**
  - **Se baja con L=1 (LOAD)**




  - **Se sube en varios lados.**
  - **Se baja en varios lados.**
  - **¿Cuántos pueden subir a la vez? ¿y bajar?**

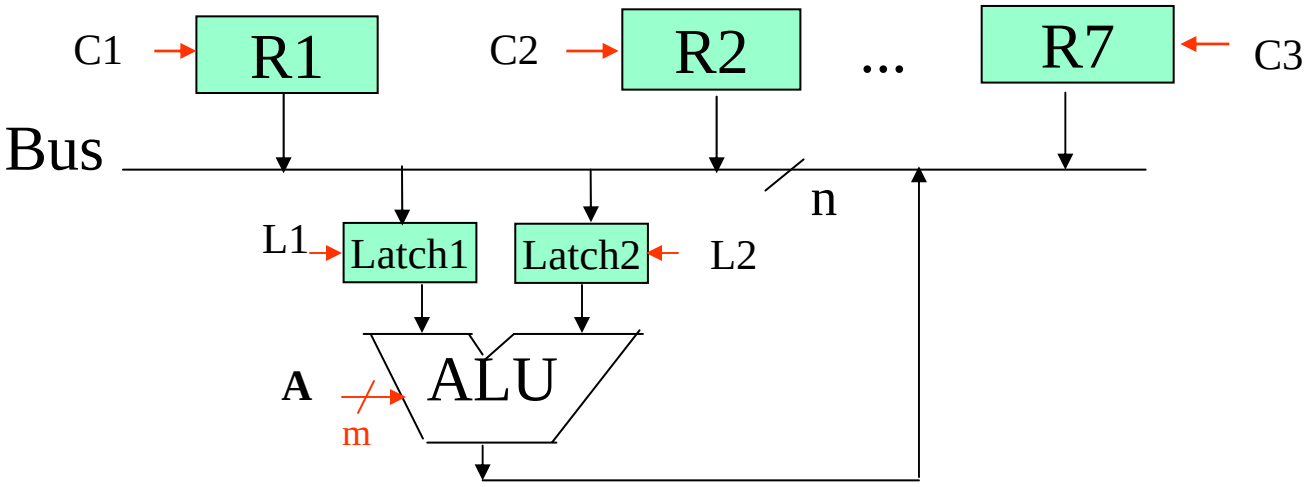
# Solución 2 – 1 Bus

- **¿Es solución?**
  - **Probar R1=R1+R2 y R3 <->R4, calcular ciclos de clock.**
  - **Ventajas y Desventajas.**
  - **El Bus es bidireccional.**



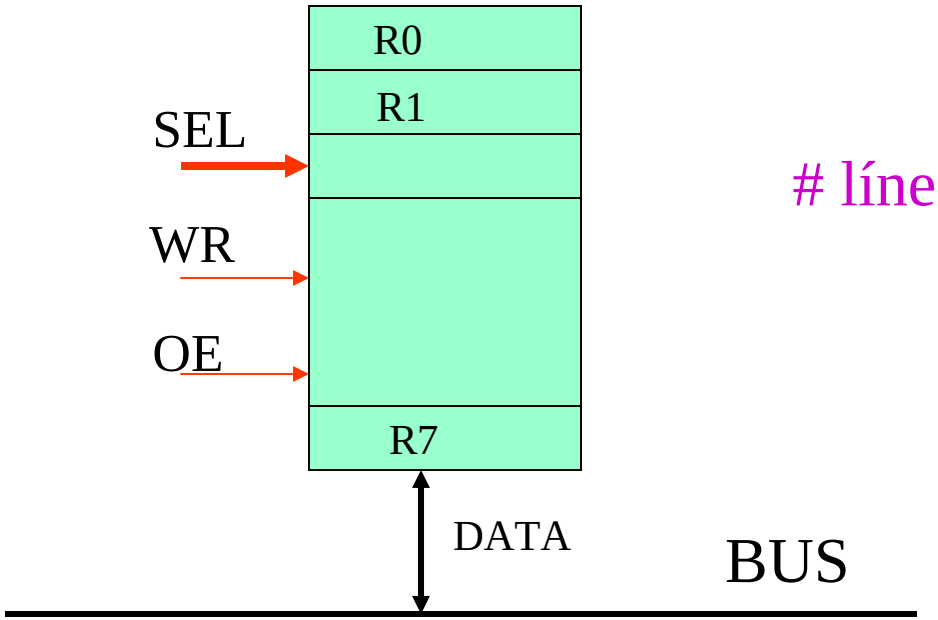
# Banco de Registros

- **Observaciones:**
  - **Solo un registro se conecta a la vez.**
  - **Muchas señales de control = O(n)... ¿se podría reducir?**
  - **Se pueden poner todos juntos en un bloque.**
  - **Con nuevas entradas de Control: SELECCIÓN, LOAD, OE.**



# Banco de Registros

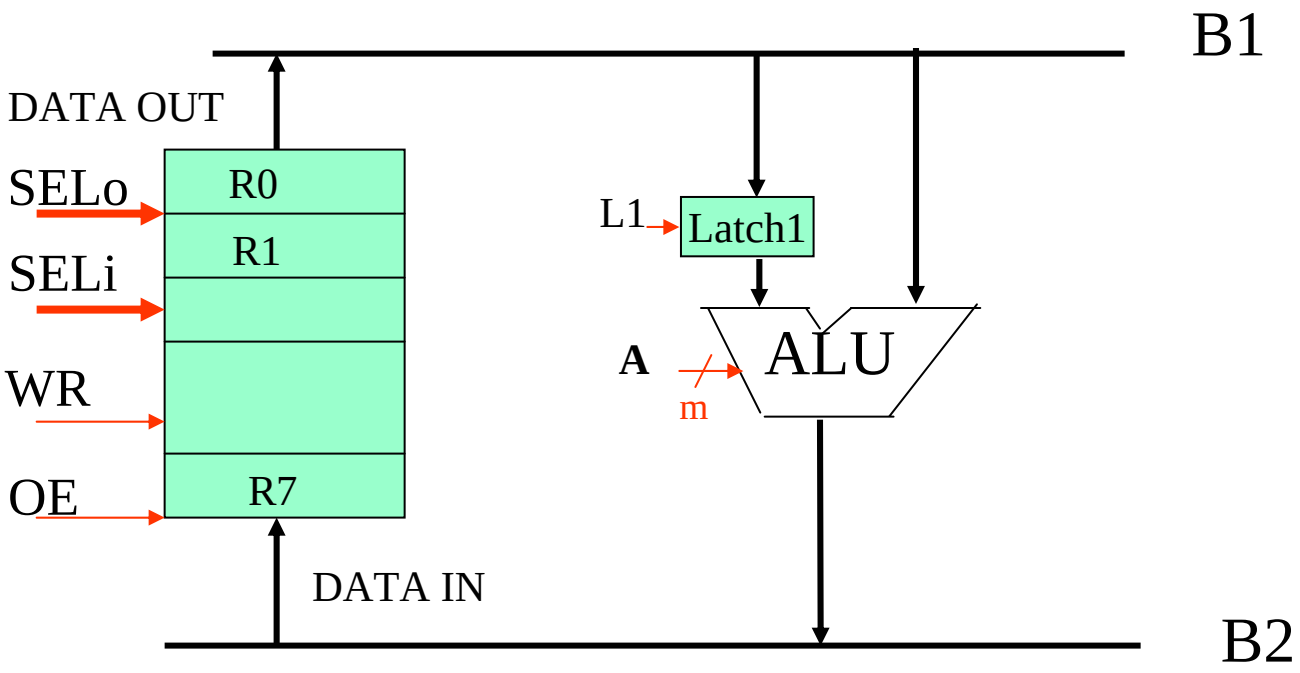
- **Entradas de Control: SELECCIÓN, WR (LOAD), OE.**
- **Ventaja: Integración – similar a una RAM.**
- **¿Cuántos bits lleva SEL?**




# líneas de control = O[log(n)]

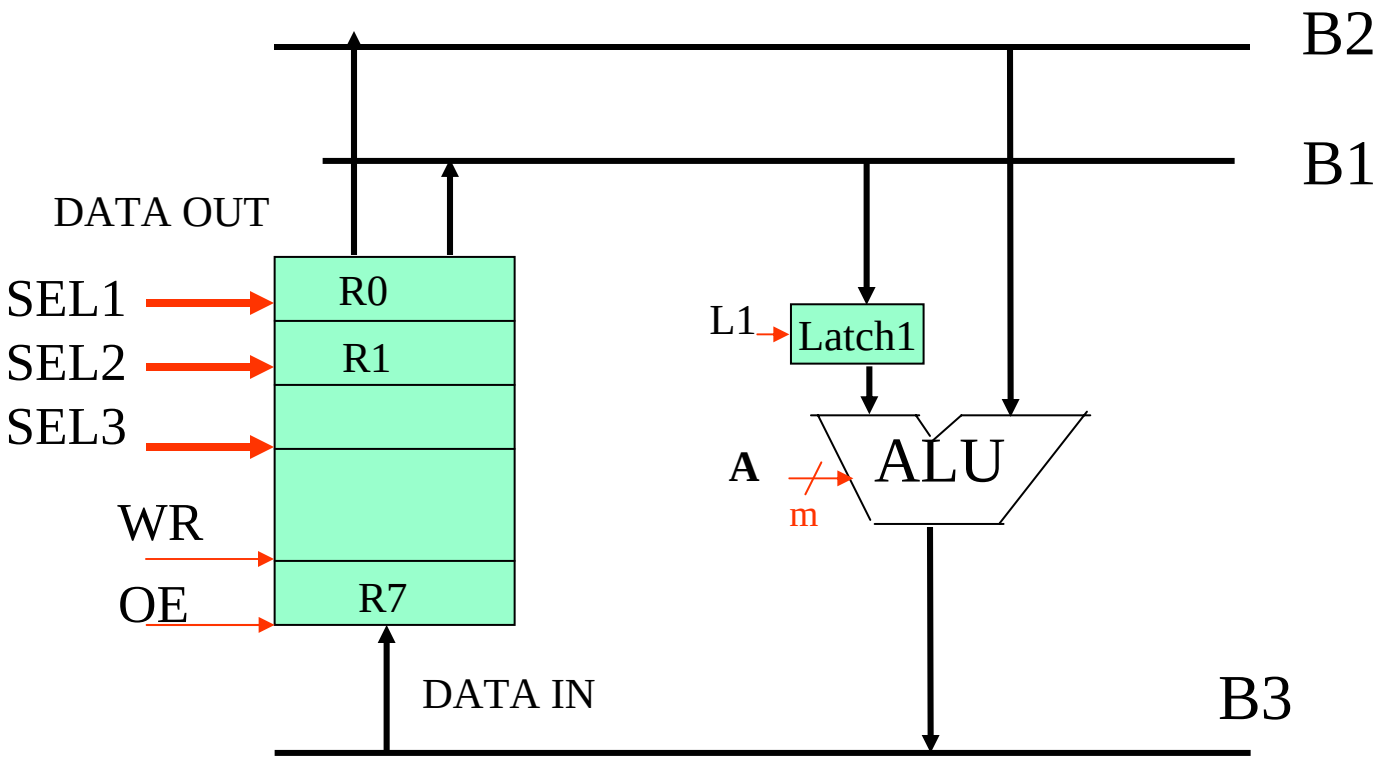
# Solución 3 – 2 Buses

- **¿Mejora la Velocidad?**
  - **Probar R1=R1+R2 y R3 <-->R4, calcular ciclos de clock.**
  - **¿Cuánto más rápido es la operación de suma?**
  - **¿Por qué no puede eliminarse el Latch L1?**

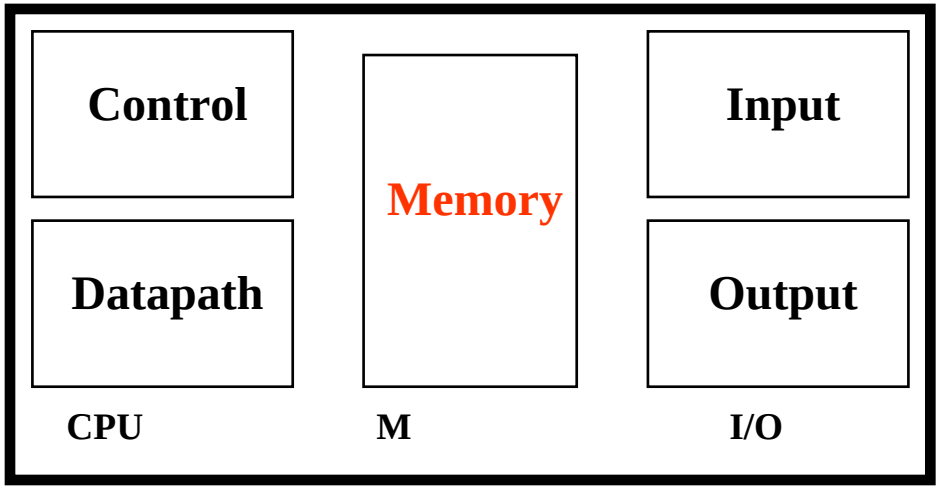


# Solución 4 – 3 Buses

- **Mejora la Velocidad**
  - **Probar R1=R1+R2 y R3 <-->R4, calcular ciclos de clock.**
  - **¿Cuánto más rápido es la operación de suma?**
  - **¿Eliminar L1?**

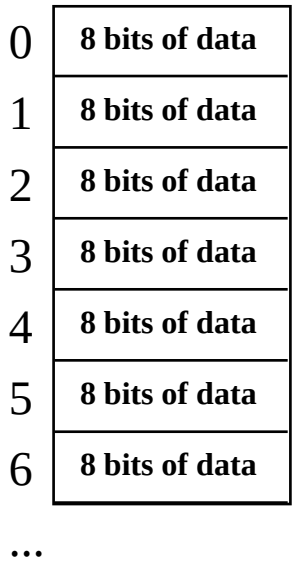


# Memorias



# Memoria - RAM

- **Funciona como una gran Tabla unidimensional.**
- **Una dirección de memoria es un índice en la tabla.**
- **“Direccionamiento de Byte" : el índice apunta a una byte en M.**

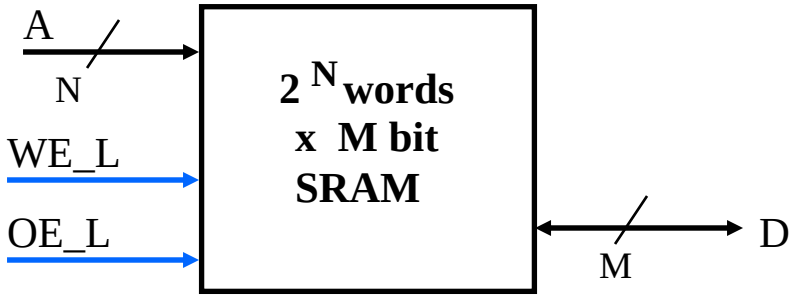



- **La celda básica es el flip-flop. Empacados en alta densidad.**

# Tecnología de Memorias

- **Acceso Aleatorio (Random):**
  - **"Random" es bueno: tiempo de acceso es el mismo para todas las celdas.**
  - **SRAM: Static Random Access Memory**
    - **Baja Densidad, alto consumo, cara, rápida**
    - **Estática: el contenido se mantiene indefinidamente con Vcc.**
  - **DRAM: Dynamic Random Access Memory**
    - **Alta Densidad, bajo consumo, barato, lento.**
    - **Dinámica: necesita ser "refrescada" regularmente.**
    - **Circuito de Refrescado – trabaja por filas completas.**
- **Memorias de acceso "no aleatorio".**
  - **Tiempo de acceso varía de lugar en lugar y de tiempo en tiempo.**
  - **Ejemplos: Discos, cintas (secuencial), CD-ROM**

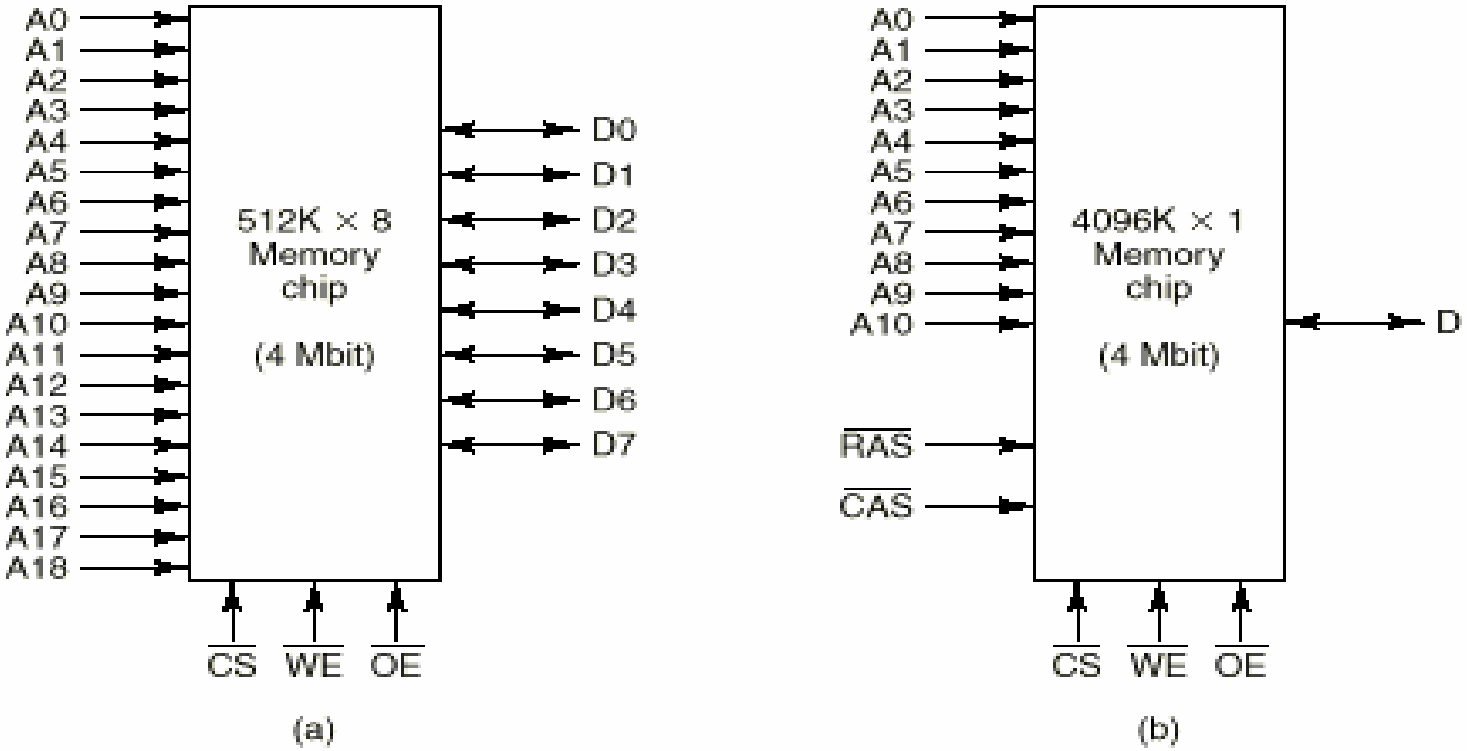
# Diagrama Lógico de una SRAM



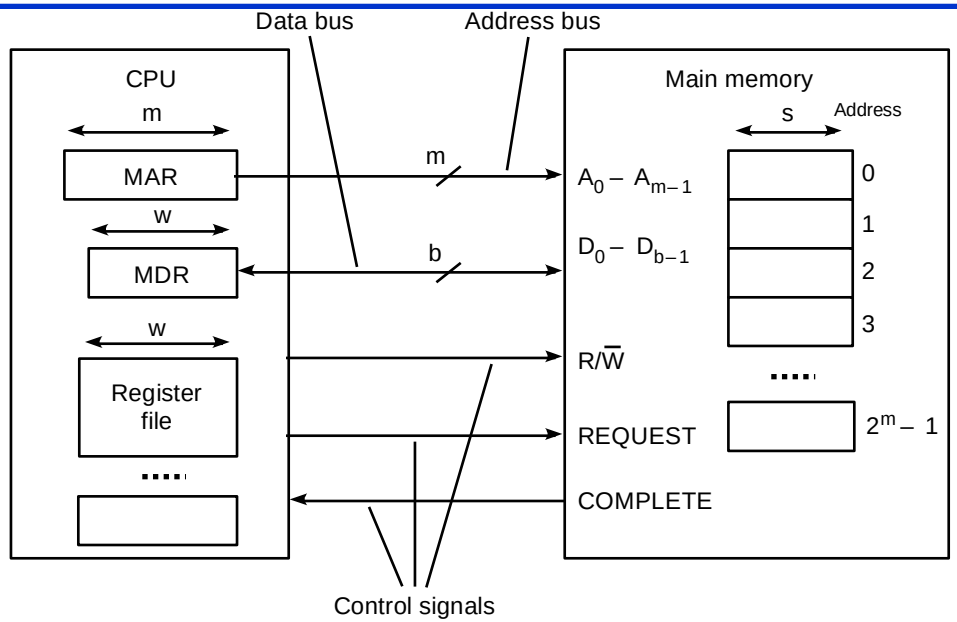

- **Señales de Control: Write Enable y Output Enable**
  - **Generalmente se activan en “Low” (WE_L, OE_L).**
  - **OE_L permite Ahorrar Pins, unificando Din y Dout.**
  - **WRITE: WE_L se activa (0), OE_L se desactiva (1)**
    - **D funciona como entrada de Datos.**
  - **READ: WE_L se desactiva (1), OE_L se activa (0)**
    - **D es la salida de datos.**
  - **INDEFINIDO: SI tanto WE_L como OE_L se activan:**
    - **Resultado no definido. ¡¡No hacerlo!!**

# Memorias: SRAM y DRAM

1. **¿Cómo se puede verificar el tamaño en KB de estos chips?**
2. **Algunas RAM unifican WE_L y OE_L en un único R/W_L**
3. **¿Para qué traen incorporados CS?**



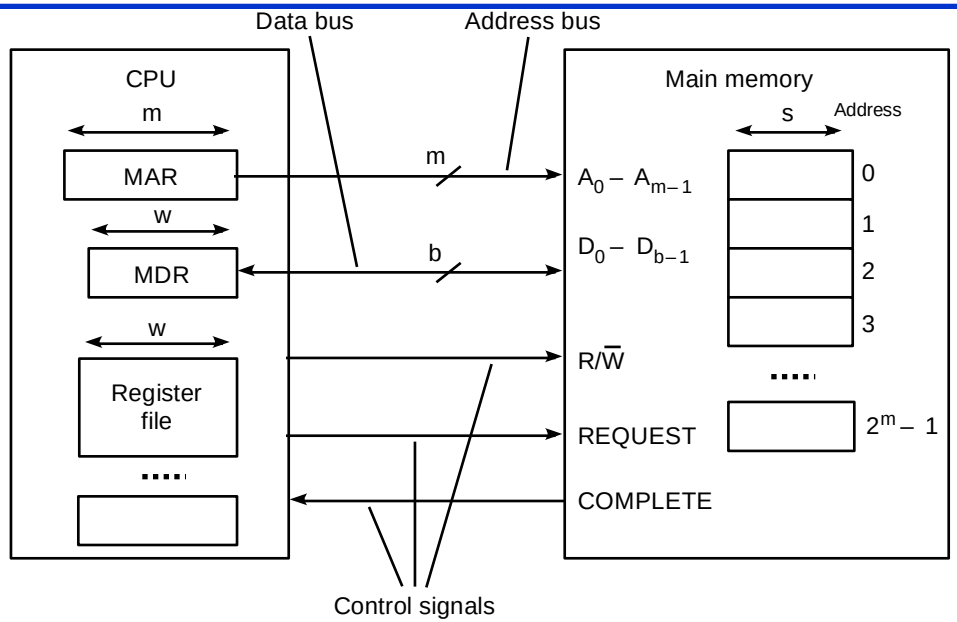
# La Interfase CPU–Memoria




1. **MAR – Memory Address Register.**
   - **Contiene la dirección de los datos a leer o escribir en M.**
   - **$2^m$ es la dimensión del "espacio de direcciones".**
2. **MDR – Memory Data Register.**
   - **Contiene los datos a escribir o que se leen de M.**
   - **w es el "ancho de palabra" del CPU. ¿w grande es bueno?**

- **DATA BUS, ADDRESS BUS y CONTROL BUS.**

# La Interfase CPU–Memoria




**Secuencia de Eventos:**

**Read:**

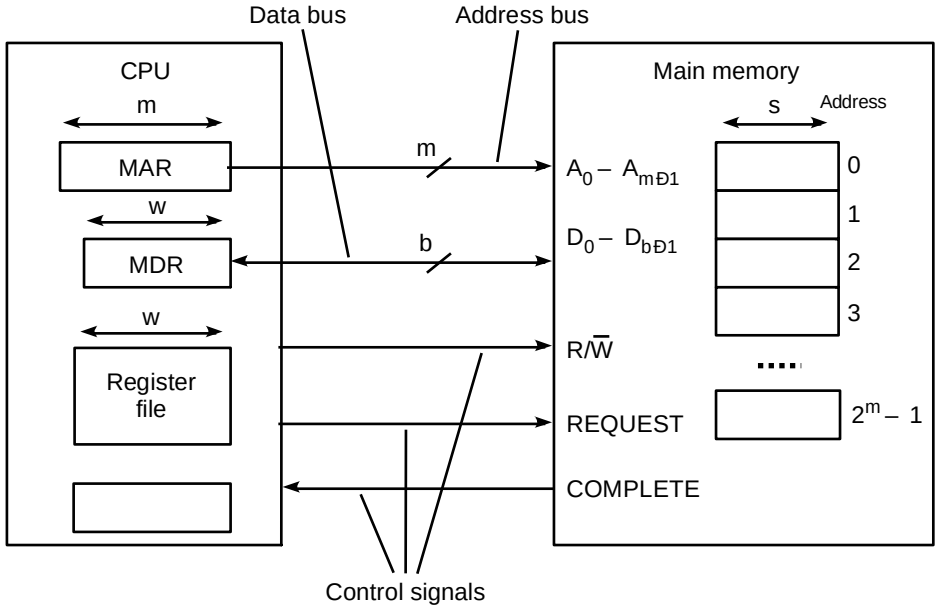
1. **CPU pone dirección en MAR, activa Read y REQUEST**
2. **Memoria envía el dato a MDR**
3. **Memoria activa COMPLETE**

**Write:**

1. **CPU carga MAR y MDR, activa Write y REQUEST**
2. **El contenido de MDR se escribe en celda indicada por MAR**
3. **Memoria activa COMPLETE**

- **¡Ojo!: No aparece CS, en la RAM no hay Request, ni Complete**

# La Interfase CPU–Memoria




**Consideraciones Adicionales:**

- **Si b < w, se requieren w/b transferencias de b-bits a Memoria.**
- **Algunos CPUs permiten leer o escribir palabras con tamaño < w**
  **Ejemplo: Intel 8088: m = 20, w = 16, s=b = 8**
  **Se pueden leer o escribir Datos de 8- y 16-bit**
- **Si la memoria es suficientemente rápida o si es predecible, entonces puede omitirse la señal COMPLETE.**
- **Algunos sistemas usan líneas OE y WE y omiten REQUEST**

# Algunas Propiedades de Memoria

| Símbolo | Definición | Intel 8088 | Intel 8086 | PowerPC 601 |
|---|---|---|---|---|
| w | Ancho de palabra del CPU | 16 bits | 16 bits | 64 bits |
| m | Bits en una dirección de M | 20 bits | 20 bits | 32 bits |
| s | Bits en menor unidad direccionable | 8 bits | 8 bits | 8 bits |
| b | Ancho del Data Bus | 8 bits | 16 bits | 64 bits |
| $2^m$ | Capac de palabras ancho s en M | $2^{20}$ words | $2^{20}$ words | $2^{32}$ words |
| $2^m$xs | Espacio de Mem. en bits | $2^{20}$ x 8 bits | $2^{20}$ x 8 bits | $2^{32}$ x 8 bits |

# Clasificación de Memoria

- **Por el Modo de Acceso desde el CPU:**
  - **Primario – se accede directamente desde CPU (RAM).**
  - **Secundario – se accede a través de operaciones de I/O (Disco, Cinta...)**
- **Por el Tiempo de Acceso a todas las celdas.**
  - **Igual para todas – RANDOM.**
  - **Diferente.**
    - **Secuencial – Cintas Magnéticas.**
    - **Mixto – Discos.**
- **Por la Volatilidad. ¿De qué sirve?**
  - **ROM – Read Only Memory.**
  - **PROM – Programable once ROM.**
  - **EPROM – Erasable PROM. (mediante UV)**
  - **EEPROM . Electrically Erasable PROM.**
- **Por la Tecnología.**
  - **SRAM.**
  - **DRAM.**
- **...**

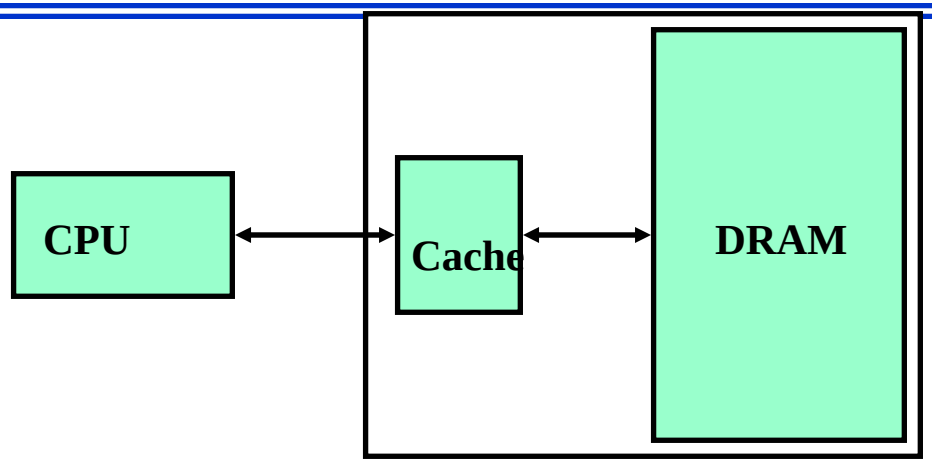
# El Principio de Localidad

- **El Principio de Localidad:**
  - **Los programas acceden a una porción relativamente pequeña del espacio de direcciones en cualquier instante de tiempo.**
- **Dos Tipos Diferentes de Localidad:**
  - **Localidad Temporal: Si un ítem (Dato o Instrucción) es accedido, tenderá a ser accedido nuevamente pronto.**
  - **Localidad Espacial: Si un item es accedido, ítems vecinos (direcciones cercanas) tenderán a ser accedidos pronto.**
- **Idea: información más usada en memoria rápida.**
- **¿Es aplicable a las Personas también?**

# Jerarquía de Memoria: Personas

- **Memoria Rápida y Limitada: el cerebro.**
- **Ayuda Memoria: Agenda.**
- **Apuntes.**
- **Libros**
- **Bibliotecas.**
- **No es transparente.**
  - **No hay un sistema automático que detecte nuestra necesidad de información y la incorpore a nuestro cerebro sin que nos demos cuenta.**

# Caché



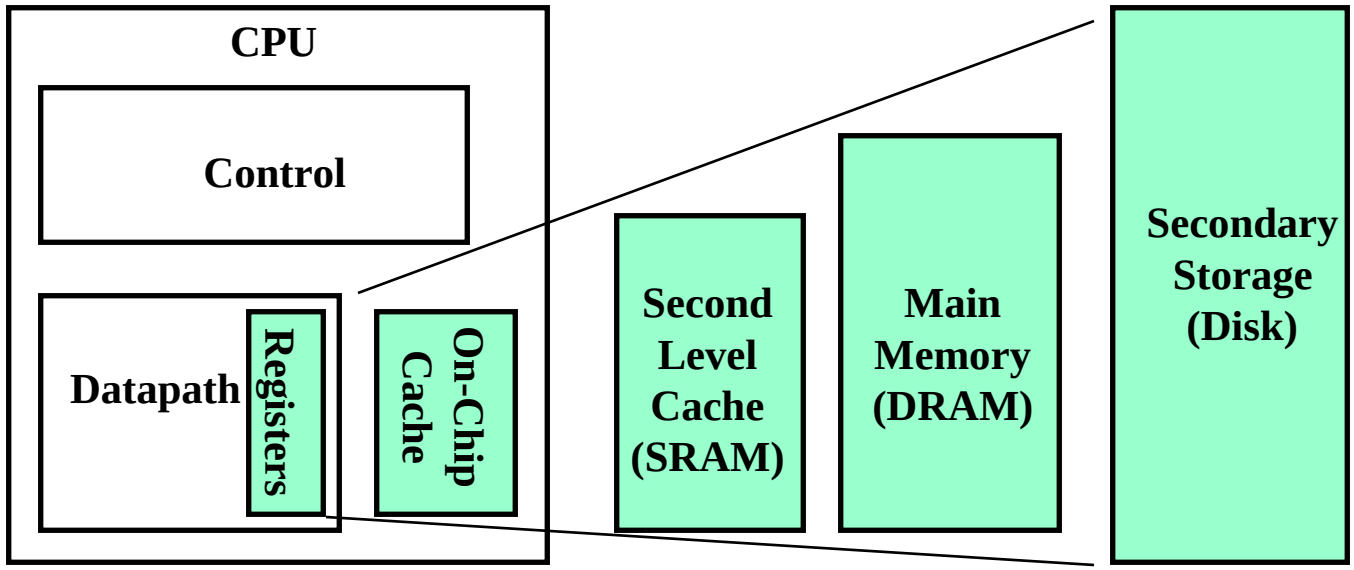

- **Motivación:**
  - **Memorias Grandes (DRAM) son lentas.**
  - **Pequeñas Memorias (SRAM) son rápidas y más caras.**
  - **Lograr un tiempo de acceso promedio pequeño:**
    - **Sirviendo la mayoría de los accesos desde la SRAM.**
  - **Reducir el ancho de banda requerido para la DRAM.**
    - **Ancho de banda = palabras transferidas segundo.**

# Cache / Memoria Virtual

- **Similar Objetivo.**
- **Cache: entre dos memorias RAM.**
- **Memoria Virtual: entre Primario (RAM) y Secundario (Disco...).**
- **Transparentes al Usuario:**
  - **Manejo por Hw. y Sistema Operativo.**

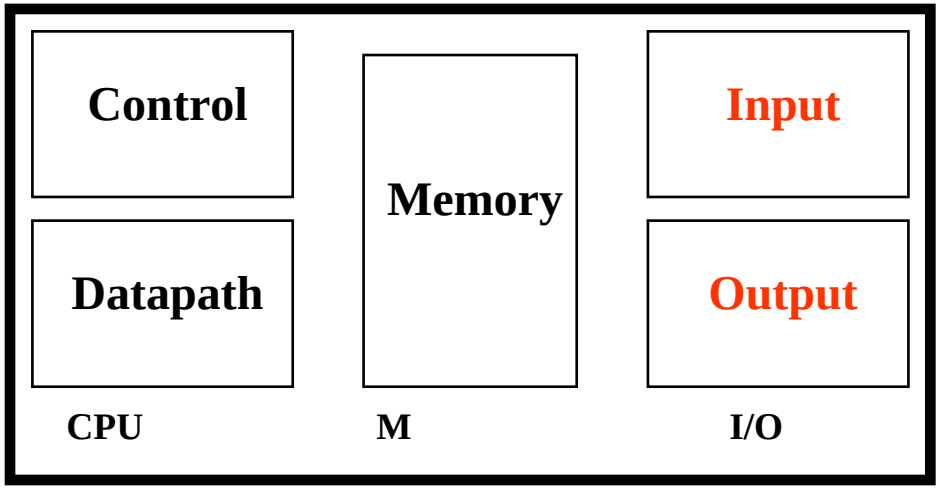
# Jerarquías de Memoria

- **Aprovechando el principio de Localidad:**
  - **Presentar al usuario todo el espacio de direcciones con un costo unitario muy cercano al de la tecnología más barata.**
  - **Brindarle acceso a la velocidad ofrecida por la tecnología más rápida.**




| | | | | | |
|---|---|---|---|---|---|
| **Velocidad (ns):** | Unos | Decenas | Cientos | 10,000,000s | (10s ms) |
| **Tamaño (bytes):** | Cientos | Kilos | Ms | Gs | |

# Entrada / Salida




**Gran Variedad de Dispositivos E/S y Velocidades de transferencia.**

# Ejemplos de Dispositivos E/S

| Dispositivo | Función | Interlocutor | Veloc. Transf. (KB/sec) |
|---|---|---|---|
| Teclado | Input | Humano | 0.01 |
| Mouse | Input | Humano | 0.02 |
| Line Printer | Output | Humano | 1.00 |
| Floppy disk | Storage | Máquina | 50.00 |
| Laser Printer | Output | Humano | 100.00 |
| Optical Disk | Storage | Máquina | 500.00 |
| Magnetic Disk | Storage | Máquina | 5,000.00 |
| Red-LAN | E/S | Máquina | 20 – 1,000.00 |
| Display Gráfico | Salida | Humano | 30,000.00 |

**¿Cómo hace un computador para manejar todo esto eficientemente?**
**¡No quedarse esperando un tecleo! ¡Hacer otras cosas mientras tanto!**

# Conceptos de E/S

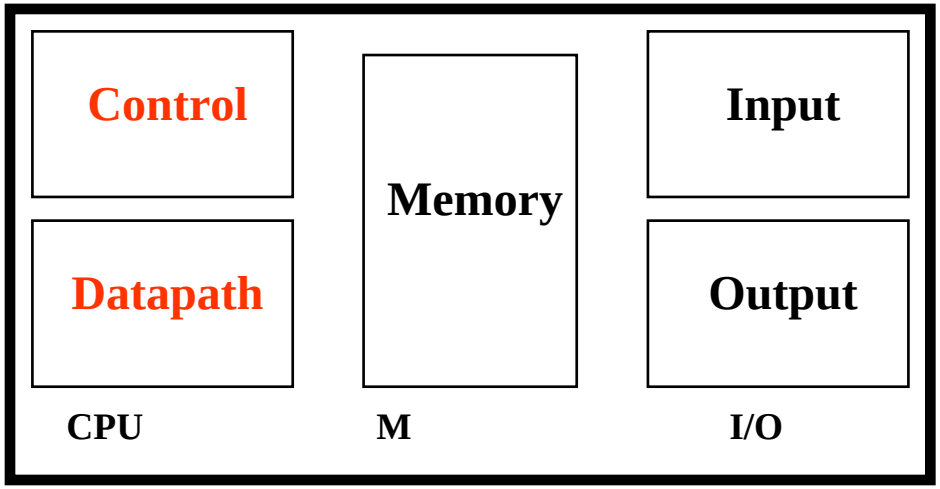
**Ej. Sistema Multitarea: Secretaria prepara Café.**

**Alternativas:**

1. **Ineficiente: Espera a que alguien pida y le sirve. No hace otra cosa (la echan)**
2. **Polling (encuestas): Cada media hora pregunta si alguien quiere café. Un reloj con alarma se lo recuerda. Si tiene otras urgencias no hace caso.**
3. **Interrupciones: Cuando alguien quiere café la llama por el interno. No hace caso si hay urgencias.**
4. **Acceso Directo (DMA): el que quiere café se sirve solo, si la cafetera se vacía suena alarma a la Secret.**
5. **Procesador Dedicado (IOP): dispensador automático de café y otras bebidas. Una vez por semana la Secretaria carga la máquina. Alarma si falla.**

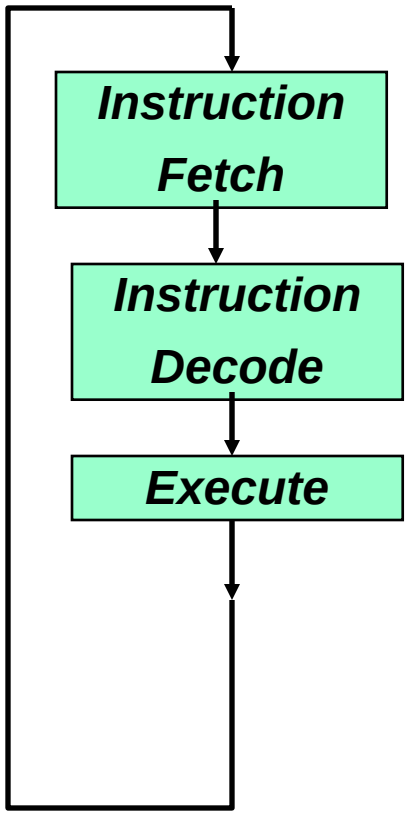
**Lo mismo vale entre un CPU y dispositivo E/S**

# Volvemos al CPU




- Registros Adicionales:
  - **PC – program Counter – dirección de la próxima instrucción.**
  - **IR – Instruction Register – contiene la instrucción actual:**

# El Ciclo de la Instrucción




**Obtener instrucción de Memoria M(PC), guardar Instrucción en IR**

**Actualizar PC**

**Determinar las acciones que se requieran**

**Ejecutar las acciones requeridas (leer o escribir en M o I/O, procesar en ALU...)**

# ¿Qué debe especificar una instrucción?

Flujo de Datos ←

- ¿Qué operación realizar? — add r0, r1, r3
  - Op code: add, load, branch, etc.
- ¿Dónde están los operando/s? — add r0, r1, r3
  - En registros del CPU, celdas de memoria, lugares de I/O o parte de una instrucción.
- ¿Dónde se guarda el resultado? — add r0, r1, r3
  - En un registro, M, I/O o Inst.
- ¿Dónde está la próxima instrucción? — add r0, r1, r3
  - En el lugar de memoria a que apunta el PC — br endloop
  - Para ej. salto – PC ←endloop

# Formato de la Instrucción

| Op-Code | Operando 1 | Operando 2 | ... | Operando n |
|---|---|---|---|---|

1. **Op-Code: Qué se hace.**
   - **n bits – hasta $2^n$ operaciones distintas.**
2. **Operandos: Con qué dato se hace.**
   - **En general desde 0 a 3 operandos.**
   - **El dato puede estar en el campo, en Memoria o en un Registro.**
   - **Modo de Direccionamiento: forma en que se especifica un dato en el campo Operando.**
3. **Puede haber más de un formato, dependiendo de Op-Code.**

# 3 Clases de Instrucciones

- **Movimiento de Datos**
  - **Tienen siempre una fuente y un destino.**
  - **Load— La fuente es memoria y el destino un registro.**
  - **Store— La fuente es un registro y el destino es memoria.**
  - **Hay casos con fuentes y destino ambos M o ambos R.**
- **Procesamiento – Instrucciones Aritméticas y Lógicas.**
  - **Procesar uno o más operandos fuentes y guardar el resultado en un lugar de destino.**
  - **Add, Sub, Shift, etc.**
- **Control de flujo de Instrucciones (Saltos)**
  - **Alterar el flujo normal de control en lugar de ejecutar la siguiente instrucción de la dirección contigua.**
  - **Br Loc, Brz Loc2,—saltos condicionales o incondicionales.**

# PSW – Program Status Word

- Registro de Código de Condición (CCR).
  - Compuesto por flags que indican cómo fue el último resultado que los cambió.
    - Carry, Negative, Overflow, Zero, Half Carry, ...
  - Sirven para implementar instrucciones de salto.
    - BZ (salte si cero), BN (salte si negativo), ...
  - Algunas arquitecturas implementan saltos sin CCR
    - Para facilitar implementación de Pipelining.
- PSW – Program Status Word.
  - Contiene los flags (CCR)
  - Contiene información de estado del CPU (más adelante)

# Modos de Direccionamiento - I

- Para escribir un programa se requiere: Algoritmo y Estructura de Datos.
- Los modos sirven para acceder a estructuras de datos de forma eficiente.
- Inmediato – para una constante.
  - El dato se escribe en el campo operando.

    Ld r1, #380H ;r1 ←380H
- Registro – para variables temporarias muy usadas.
  - En el operando figura el número de un registro en el que está el dato.

    Si r2 contiene 3 y r3 contiene 4.

    Add r1, r2, r3 ;r1 ←7

# Modos de Direccionamiento - II

- Directo – variables globales.
  - En el operando se escribe la dirección del dato en M.

    Ej. Si M(100H)=5

    Ld r1, (100H)  ; r1←5
- Indexado – punteros a tablas.
  - En el operando se escribe un registro r y un valor llamado desplazamiento d.
  - El dato se encuentra en M(#d+r).

    Ej. Si r2 contiene 100H y si M(130H)=5

    Ld r1, 30(r2)  ; r1←5.

# Modos Comunes de Direccionamiento

**Ejemplo: Identificar los modos del SRC (más adelante)**

**Registro Destino**

| Nombre del Modo | Sintaxis Assembler | RTN significado | Uso |
|---|---|---|---|
| Registro | Ra | R[t] ← R[a] | Variable Temporaria |
| Registro indirecto | (Ra) | R[t] ← M[R[a]] | Puntero |
| Inmediato | #X | R[t] ← X | Constante |
| Directo / Absoluto | X | R[t] ← M[X] | Variables Globales |
| Indirecto | (X) | R[t] ← M[ M[X] ] | Puntero Variable |
| Indexado, base, o desplazamiento | X(Ra) | R[t] ← M[X + R[a]] | Tablas, vectores |
| Relativo | X(PC) | R[t] ← M[X + PC] | Valores almacenados en el programa |
| Autoincremento | (Ra)+ | R[t] ← M[R[a]]; R[a] ← R[a] + 1 | Acceso |
| Autodecremento | - (Ra) | R[a] ← R[a] - 1; R[t] ← M[R[a]] | Secuencial a tablas |

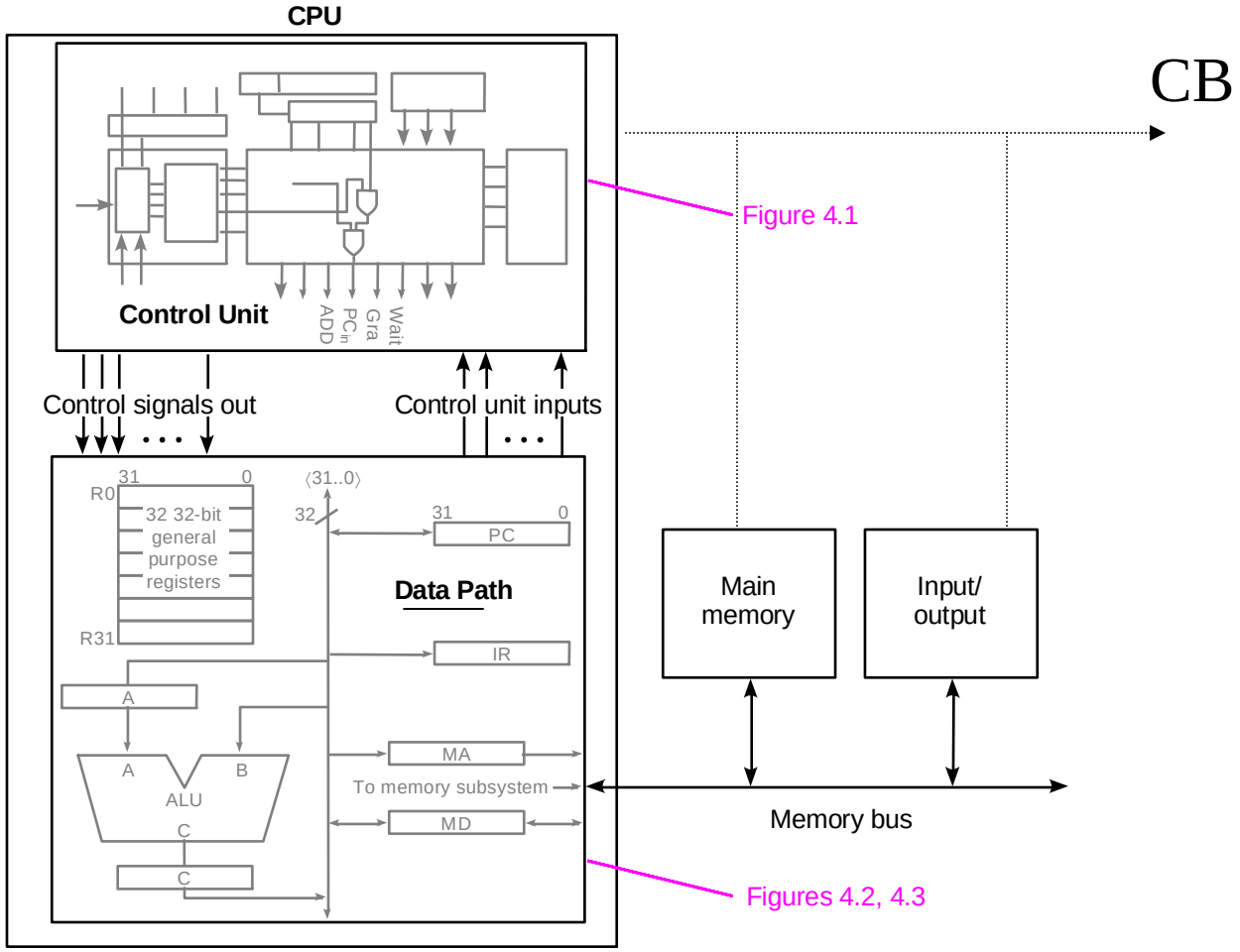
# ¿Cuáles son las componentes de un ISA?

- **Conocido como el *Modelo del Programador de la máquina*.**
- **Celdas de Almacenamiento**
  - **Registros de propósito general y especial en el CPU.**
  - **Muchas celdas de propósito general de igual tamaño en Memoria.**
  - **Almacenamiento relacionado con dispositivos I/O.**
- **El Set de Instrucciones de la Máquina.**
  - **Es el repertorio completo de las operaciones de la máquina.**
  - **Emplea celdas de almacenamiento, formatos y resultados del ciclo de la Instrucción.**
    - **Ej. Transferencias de Registros.**
- **Formato de la Instrucción**
  - **Tamaño y significado de los diferentes campos de la Instrucción.**

# Arquitectura del Set de Instrucciones (ISA)

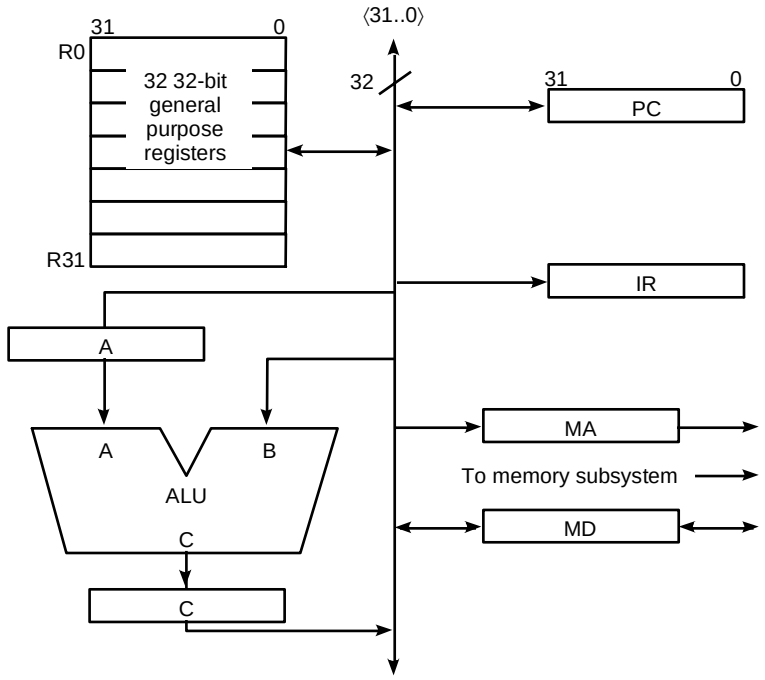
- **ISA = Información que se requiere para escribir programas en el CPU.**
- **Presentaremos un CPU y su trayectoria de datos.**
- **Presentaremos algunas Instrucciones.**
- **Veremos todas las operaciones que se realizan para ejecutar un programa.**

# Ej: Diagrama de Bloques CPU de 1 Bus: SRC - 1

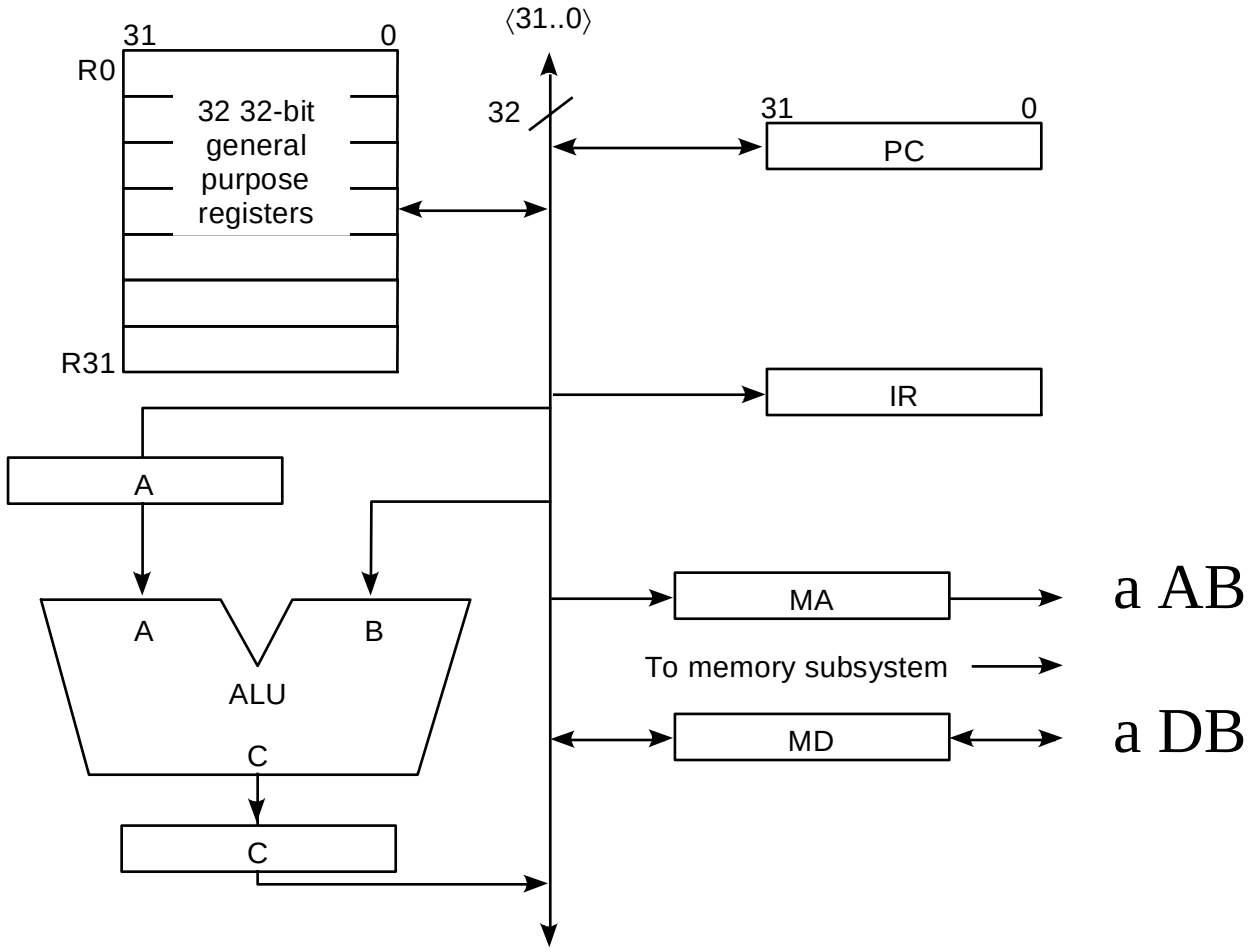


# Comentarios Adicionales

- Un solo Bus permite solo una transferencia de Registro a la vez.
- El primer operando de la ALU siempre en A, resultado en C
- Segundo Operando de la ALU proviene siempre del Bus
- Palabras de Mem de 32 bits.
- Direccionamiento a Bytes en M.
- MAR tiene 32 bits.
- ¿Espacio de direcciones?
- 32 Registros de uso General R(0...31)
- Tiempo de Acceso a Mem = 1T



# Trayectoria de Datos

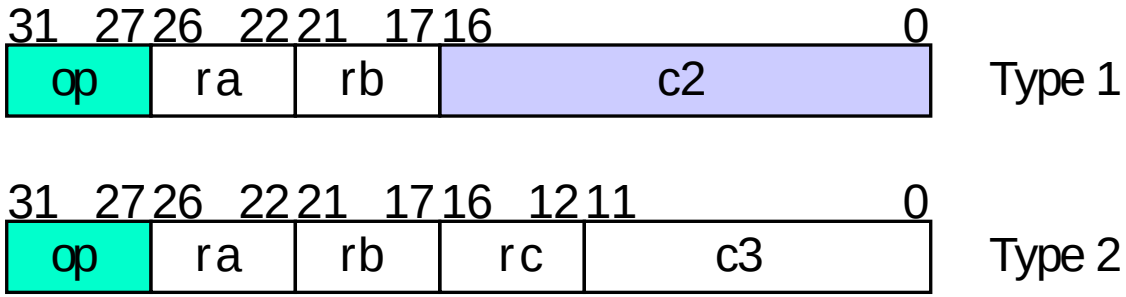


# Descripción Formal de Instrucciones

- RTN – Register Transfer Notation o RTL.
  - **Descripción Abstracta – dice qué se hace pero no cómo se implementa.**
  - **Descripción Concreta – dice cómo se implementa.**
  - **Se trata de diseño – puede haber distintas implementaciones, con ventajas y desventajas.**
  - **Debe haber una única descripción abstracta.**
  - **Puede haber múltiples descripciones concretas.**
  - **La trayectoria de datos no forma parte del ISA**.

# Instrucciones SRC-1

**Formatos**:




**Abstract RTN: (IR ← M[PC]: PC ← PC + 4; instruction_execution);**
**instruction_execution := ( • • •**
**add (:= op= 12) → R[ra] ← R[rb] + R[rc]:**
**ld (:= op= 1) → R[ra] ← M[disp] :**
**st (:= op= 3) → M[disp] ← R[ra] : ...)**

**donde**

**disp⟨31..0⟩ := ((rb=0) → c2⟨16..0⟩ {sign ext.} :**
**(rb≠0) → R[rb] + c2⟨16..0⟩ {sign extend, 2's comp.} ) :**

**Extender signo significa repetir el MSB.**

**Implementación: R(0)=0 siempre y todo se simplifica.**

# Observaciones del ISA - SRC

- **Tipo LOAD/STORE.**
  - **Sólo las instrucciones Load y Store acceden a M.**
  - **Se opera con registros.**
  - **Objetivo: ancho de banda de M menos exigido.**
  - **Las instrucciones entran siempre en una palabra.**
  - **Pocas instrucciones y simples.**
  - **RISC – "Reduced Instruction Set Computer"**
- ¿Cuáles son los modos de direccionamiento?
  - Inmediato, Directo, Indexado, Registro Indirecto.
- SRC-1 = Simple Risc Computer con 1 Bus.

# RTN Concretas para ADD

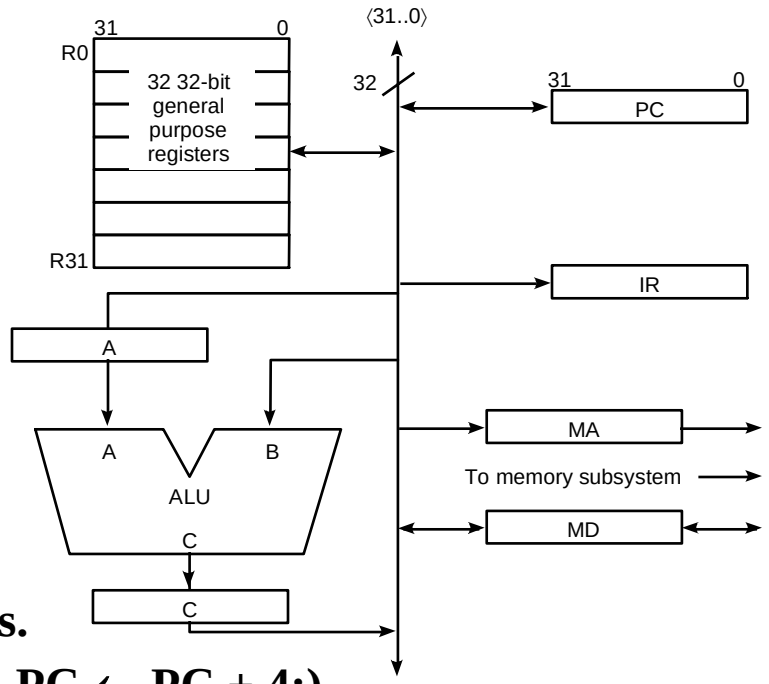
**Abstract RTN: (IR ← M[PC]: PC ← PC + 4; instruction_execution);**
**instruction_execution := ( • • •**
**add (:= op= 12) → R[ra] ← R[rb] + R[rc]:**

**RTN concreta para add**

| Paso | RTN |
|---|---|
| T0 | MAR ← PC: C ← PC + 4; |
| T1 | MDR ← M[MAR]: PC ← C; |
| T2 | IR ← MDR; |
| T3 | A ← R[rb]; |
| T4 | C ← A + R[rc]; |
| T5 | R[ra] ← C; |

IF

IEx.




- **FETCH (IF) es igual para todas las instrucciones.**
- **En T0 se ejecutan parte de 2 RTs (IR ← M[PC]: PC ← PC + 4;)**
- **La ejecución de una sola RT abstracta en ADD toma 3 RTs concretas (T3, T4, T5)**

# RTN Concretas para Ld y St

**Ld (:= op= 1) → R[ra] ← M[disp] :**
**St (:= op= 3) → M[disp] ← R[ra] :**
**donde**
**disp⟨31..0⟩ := ((rb=0) → c2⟨16..0⟩ {sign ext.} :**
**(rb≠0) → R[rb] + c2⟨16..0⟩ {sign extend, 2's comp.} ) :**

## Instrucciones Ld and St (load/store register from memory)

**16@IR(16)# significa "repetir 16 veces el bit 16 de IR y concatenar con..."**

| Paso | RTN para Ld | RTN para St | |
|---|---|---|---|
| T0–T2 | Instruction fetch | | |
| T3 | A ← (rb = 0 → 0: rb ≠ 0 → R[rb]); | | R(0):=0 siempre |
| T4 | C ← A + (16@IR⟨16⟩#IR⟨15..0⟩); | | ext. signo |
| T5 | MAR ← C; | | |
| T6 | MDR ← M[MAR]; | MDR ← R[ra]; | |
| T7 | R[ra] ← MDR; | M[MAR] ← MDR; | |

# Ejemplo de Programa

- Datos
  - **00000100H** **2** **(valor de la palabra)**
  - **00000104H** **3**
  - **00000108H** **(lugar para resultado)**
- Programa
  - **00000000H** **ld** **r1, 0, 0100H**
  - **00000004H** **ld** **r2, 0, 0104H**
  - **00000008H** **add** **r3, r2, r1**
  - **0000000CH** **st** **r3, 0, 0108H**
- Programa y Datos están en Memoria.

# Ejecución: *Ld r1, 0, 0100H*

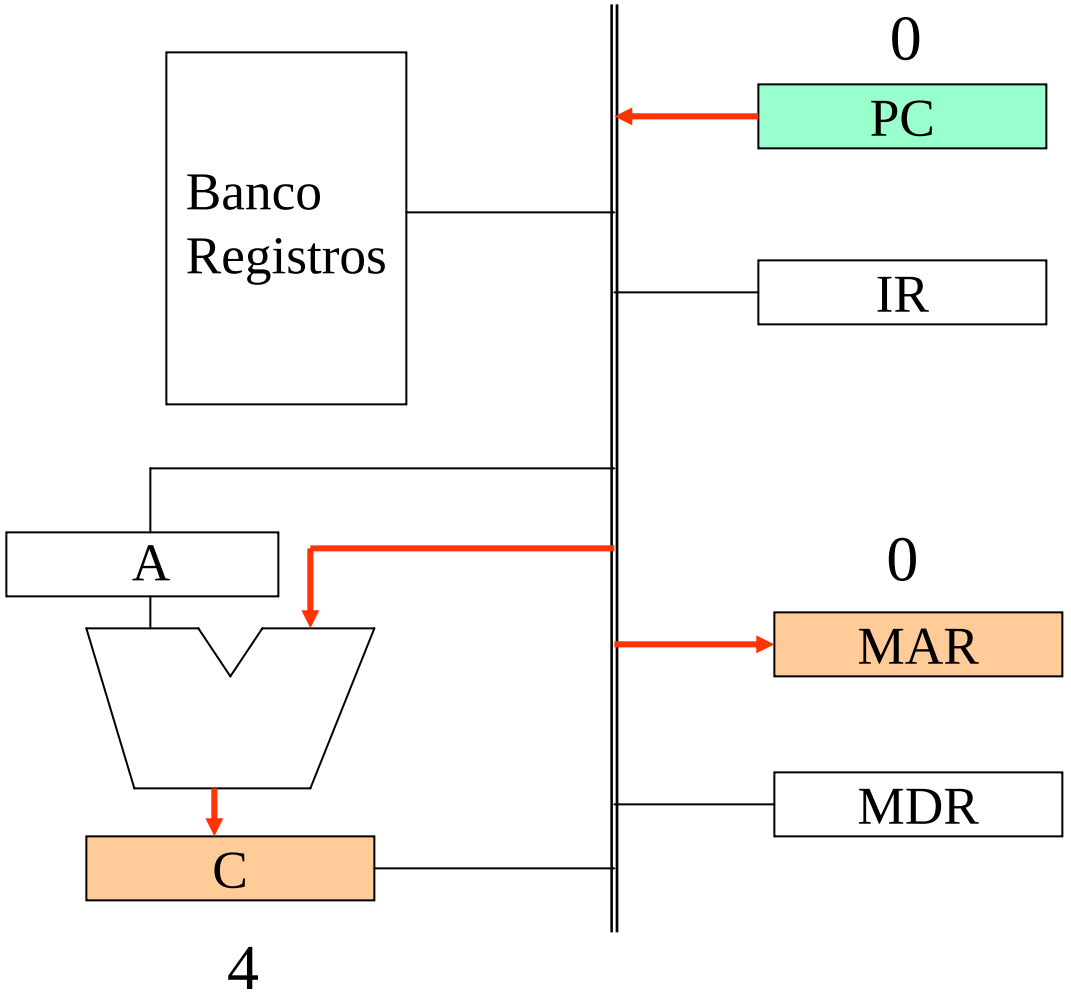
- Inicialmente PC=00000000H

| Paso | RTN para ld | |
|---|---|---|
| **Fetch** | | |
| T0 | MAR ← PC: C ← PC + 4; | MAR=0, C=4 |
| T1 | MDR ← M[MAR]: PC ← C ; | MDR=M(0)=Ld, PC=4 |
| T2 | IR ← MDR; | IR=Ld |
| **Ejecución** | | |
| T3 | A ← (rb = 0 → 0: rb ≠ 0 → R[rb]); | A=0 |
| T4 | C ← A + (16@IR⟨16⟩#IR⟨15..0⟩); | C=00000100H |
| T5 | MAR ← C; | MAR=00000100H |
| T6 | MDR ← M[MAR]; | MDR=M(100H)=2 |
| T7 | R[ra] ← MDR; | R(1)=2 |

- Veamos Paso a Paso.

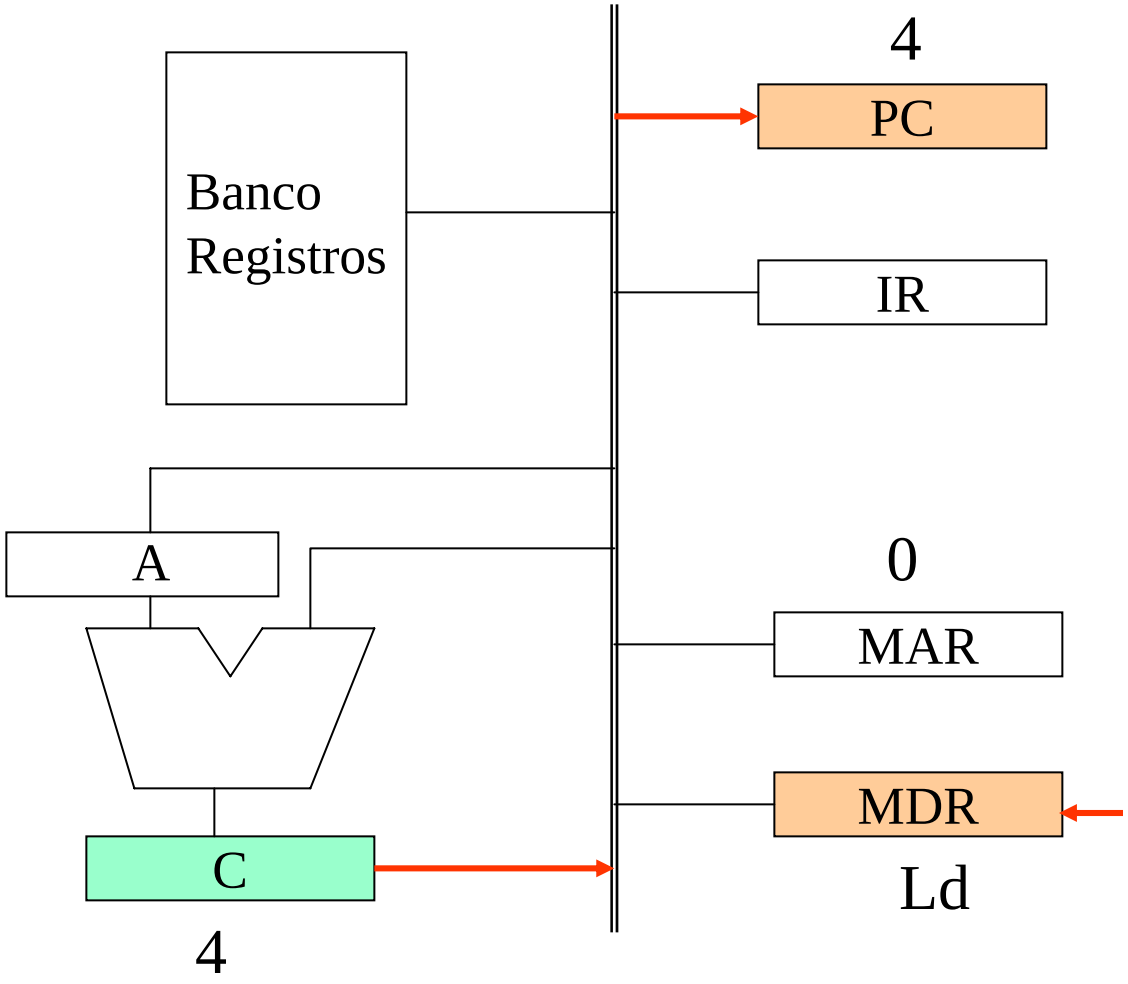
# Fetch: *Ld r1, 0, 0100H*

**T0: MAR ← PC: C ← PC + 4;**



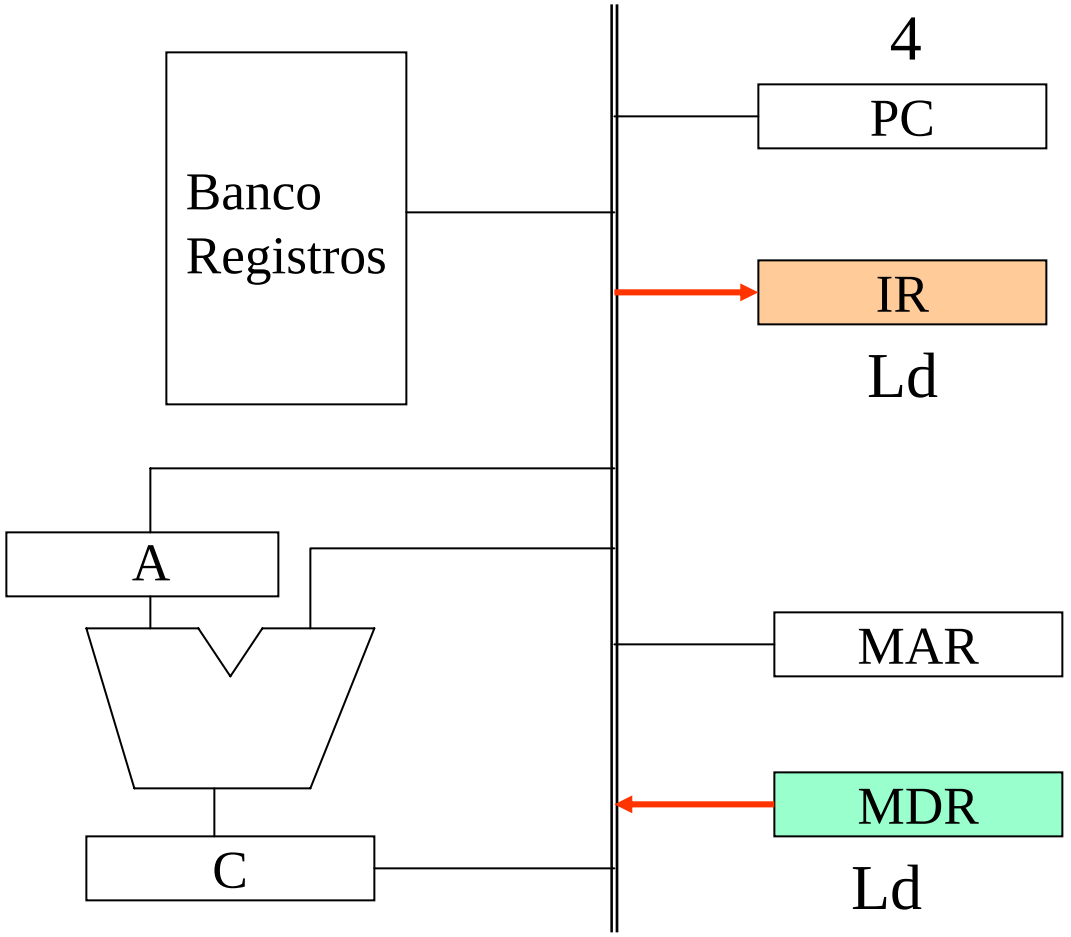
# Fetch: *Ld r1, 0, 0100H*

**T0: MAR ← PC: C ← PC + 4;**
**T1: MDR ← M[MAR]: PC ←C;**



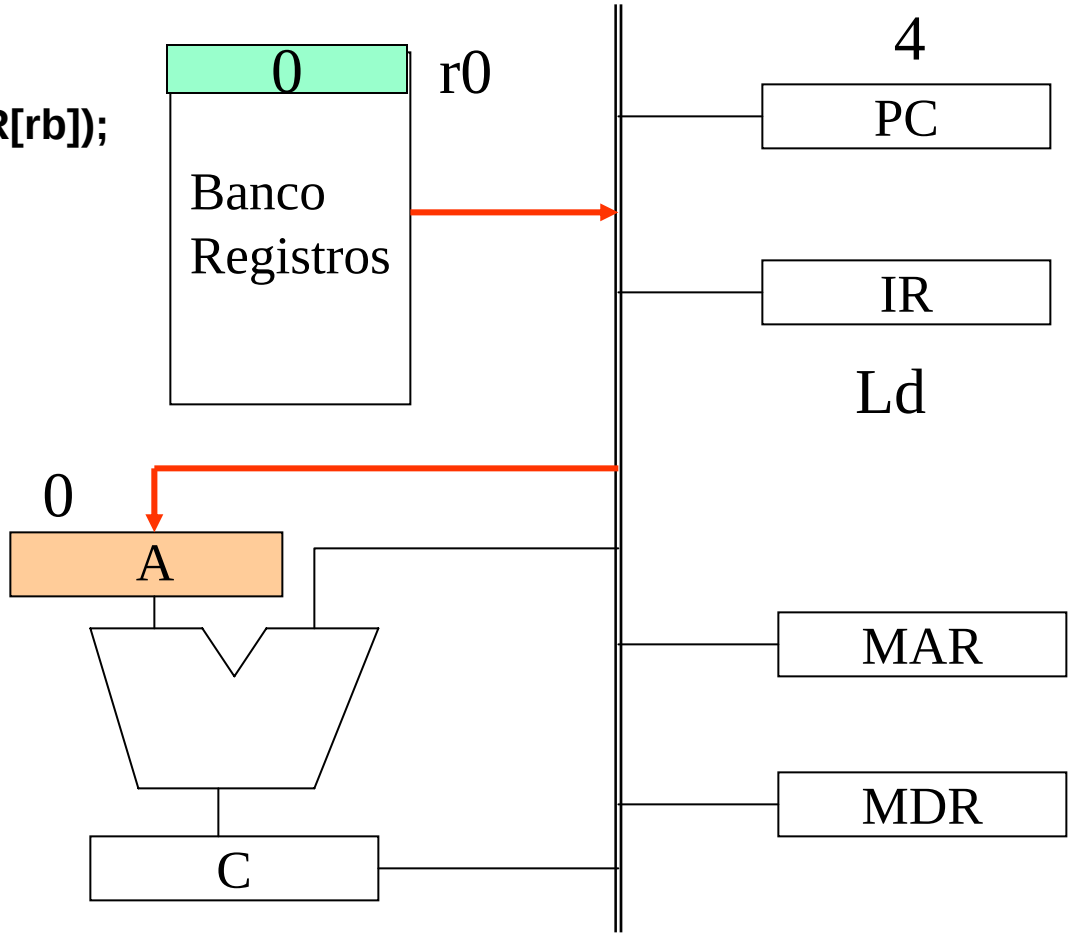
# Fetch: *Ld r1, 0, 0100H*

**T0: MAR ← PC: C ← PC + 4;**
**T1: MDR ← M[MAR]: PC ←C;**
**T2: IR ← MDR;**



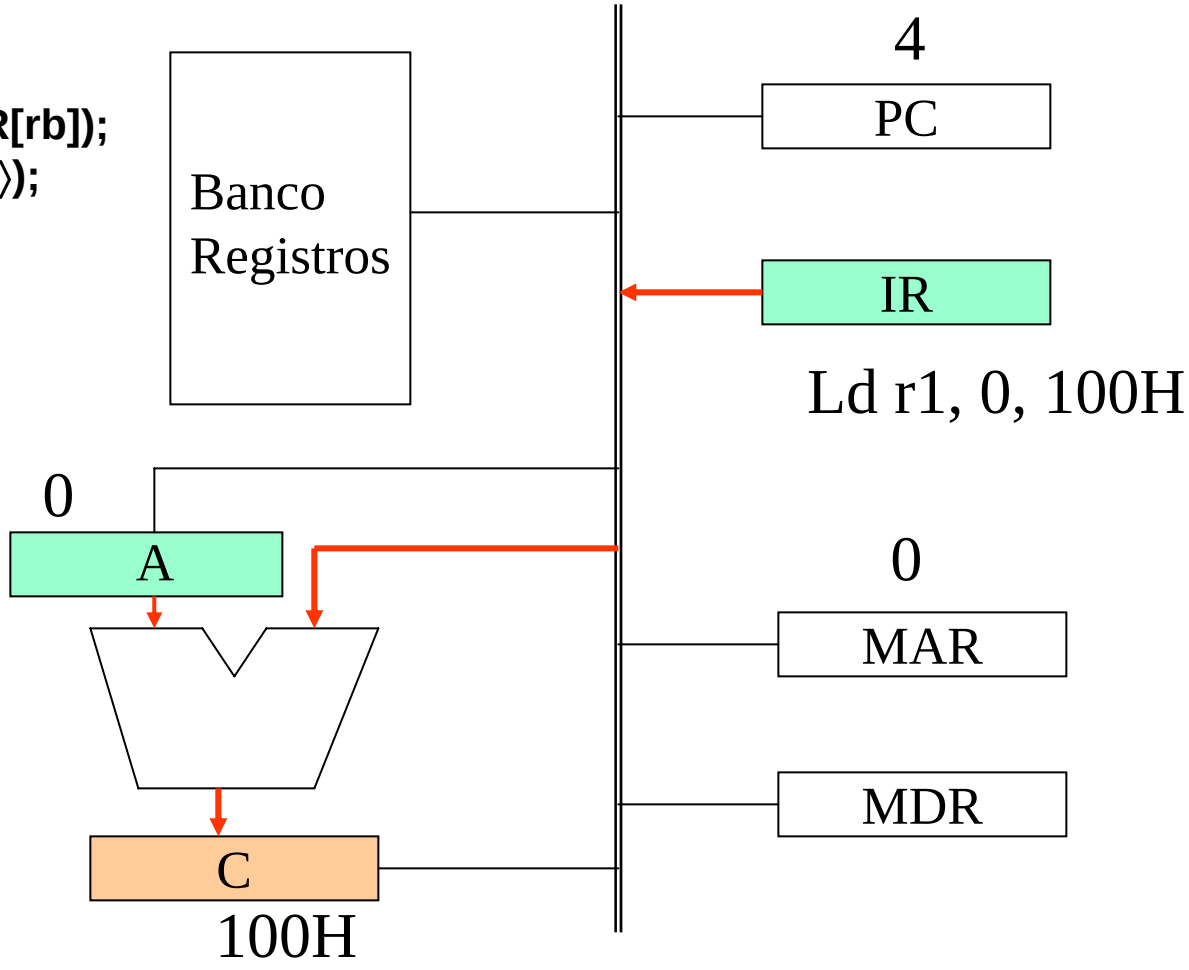
# Ejecución: *Ld r1, 0, 0100H*

**T0: MAR ← PC: C ← PC + 4;**
**T1: MDR ← M[MAR]: PC ←C;**
**T2: IR ← MDR;**
**T3: A ← (rb = 0 → 0: rb ≠ 0 → R[rb]);**



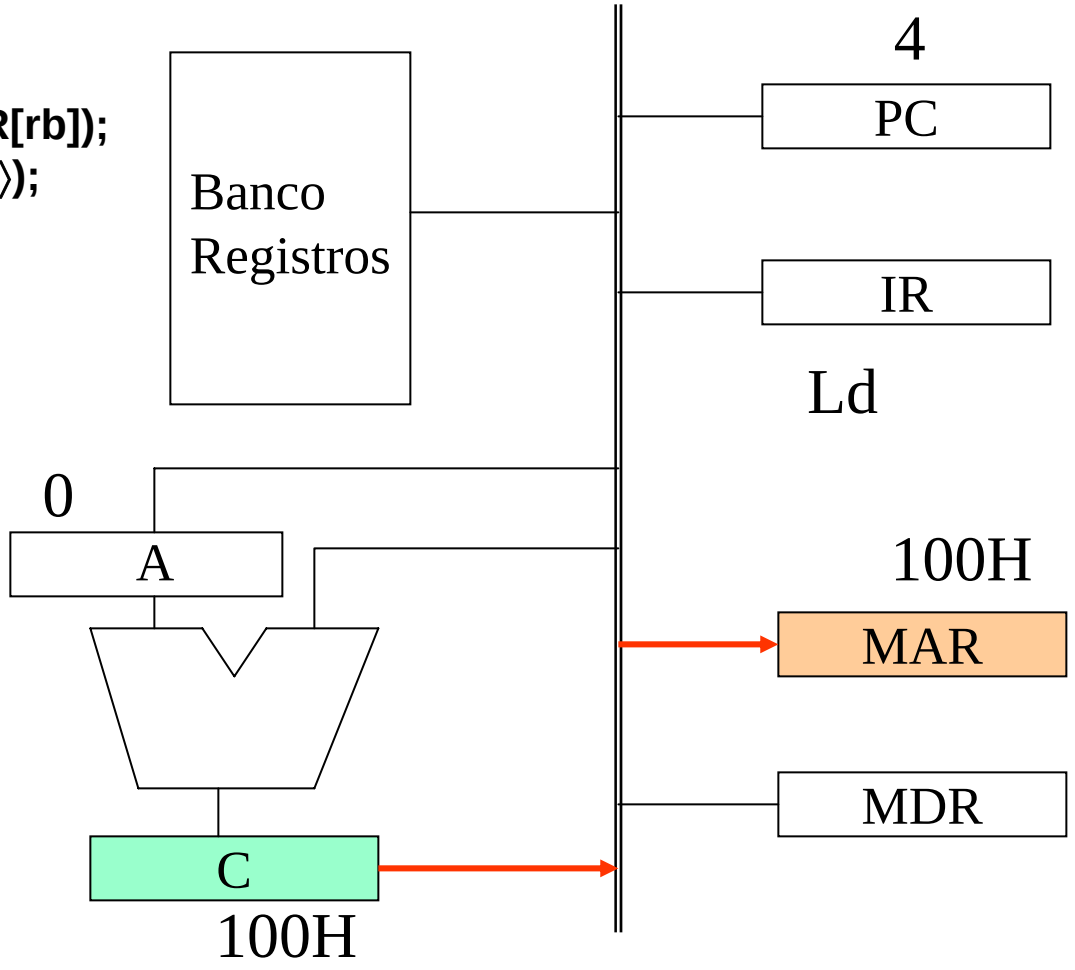
# Ejecución: *Ld r1, 0, 0100H*

**T0:** MAR ← PC: C ← PC + 4;
**T1:** MDR ← M[MAR]: PC ←C;
**T2:** IR ← MDR;
**T3:** A ← (rb = 0 → 0: rb ≠ 0 → R[rb]);
**T4:** C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);



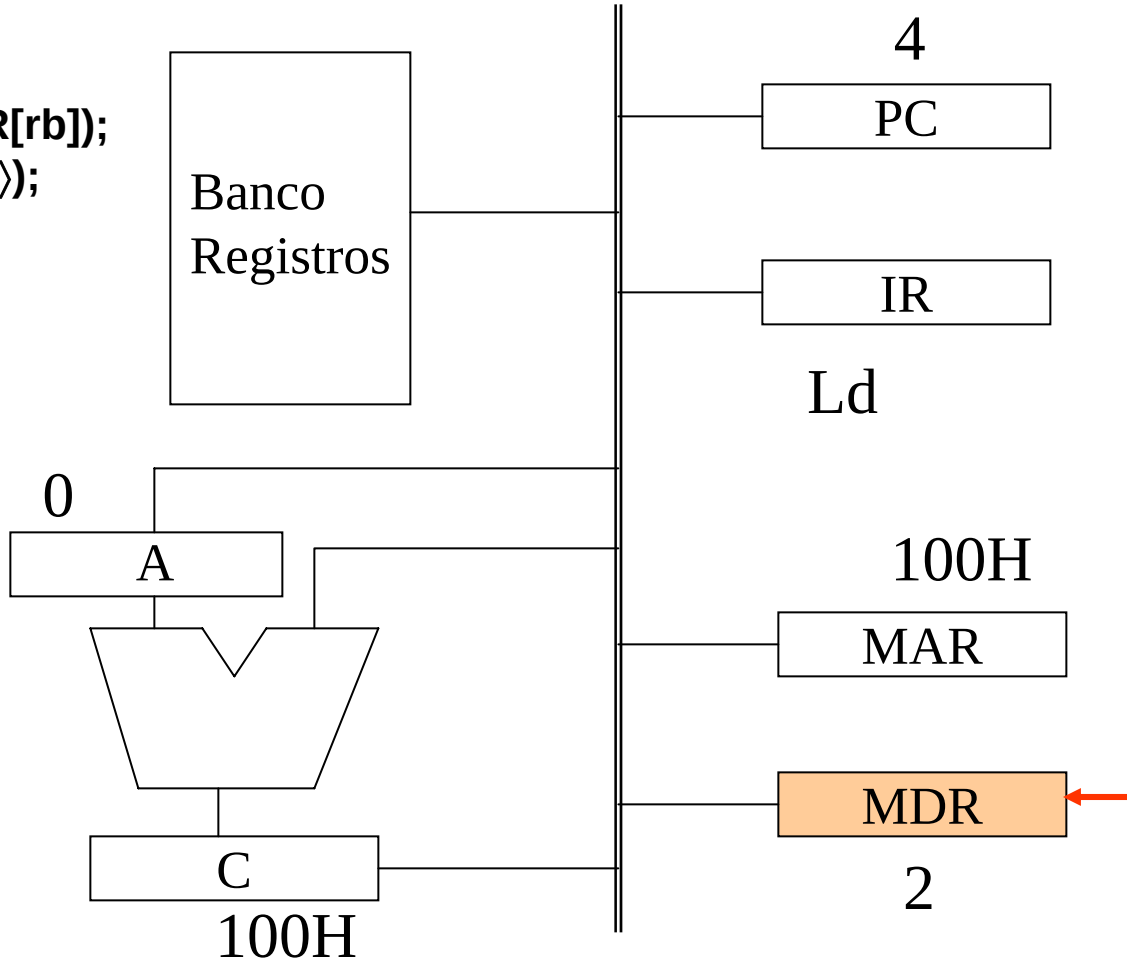
# Ejecución: *Ld r1, 0, 0100H*

**T0: MAR ← PC: C ← PC + 4;**
**T1: MDR ← M[MAR]: PC ←C;**
**T2: IR ← MDR;**
**T3: A ← (rb = 0 → 0: rb ≠ 0 → R[rb]);**
**T4: C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);**
**T5: MAR ← C;**



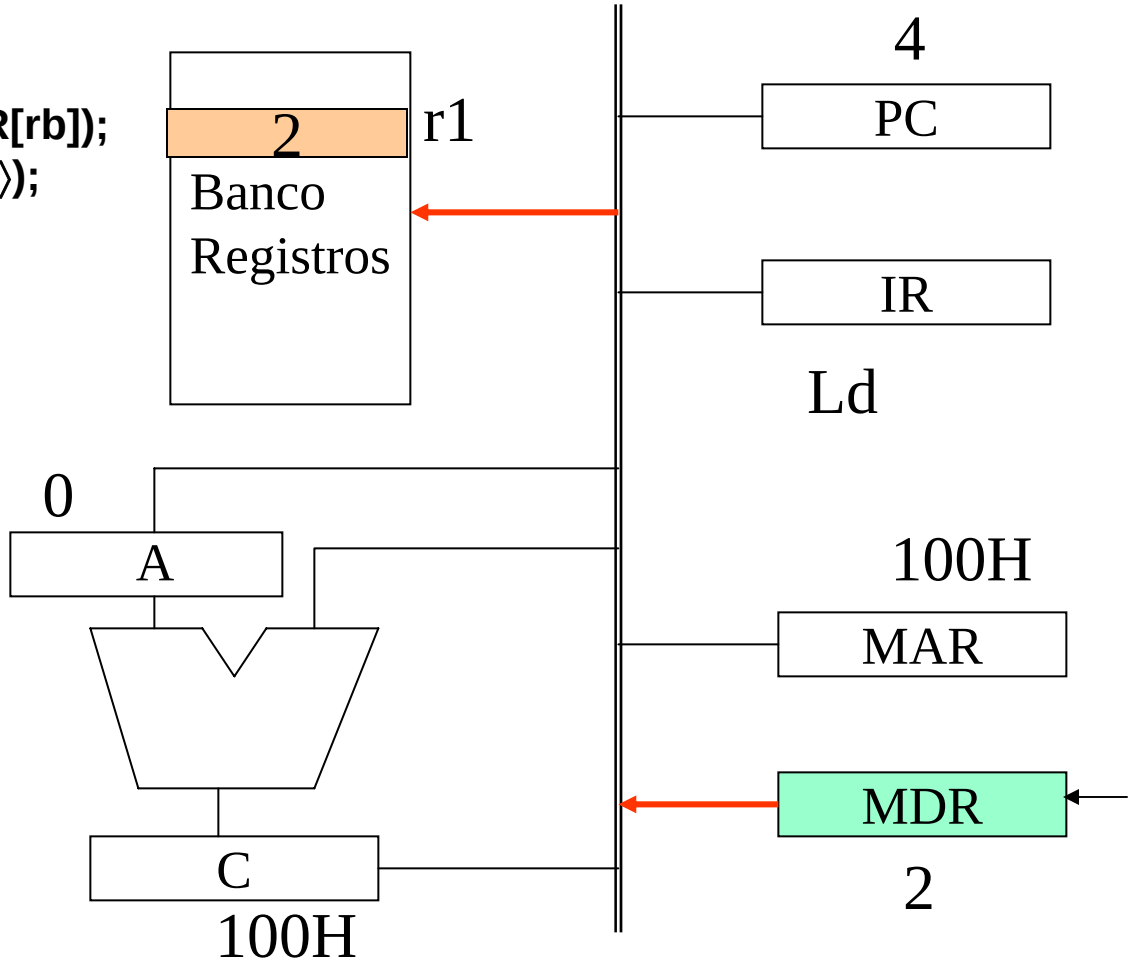
# Ejecución: *Ld r1, 0, 0100H*

**T0: MAR ← PC: C ← PC + 4;**
**T1: MDR ← M[MAR]: PC ←C;**
**T2: IR ← MDR;**
**T3: A ← (rb = 0 → 0: rb ≠ 0 → R[rb]);**
**T4: C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);**
**T5: MAR ← C;**
**T6: MDR ← M[MAR];**



# Ejecución: *Ld r1, 0, 0100H*

**T0:** MAR ← PC: C ← PC + 4;
**T1:** MDR ← M[MAR]: PC ←C;
**T2:** IR ← MDR;
**T3:** A ← (rb = 0 → 0: rb ≠ 0 → R[rb]);
**T4:** C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);
**T5:** MAR ← C;
**T6:** MDR ← M[MAR];
**T7:** R[ra] ← MDR;



# Ejecución: *Ld r2, 0, 0104H*

- PC = 00000004H, r1=2

| Paso | RTN para ld | |
|---|---|---|
| Fetch | Ídem anterior – para vagos creativos | |
| T8 | MAR ← PC: C ← PC + 4; | MAR=4, C=8 |
| T9 | MDR ← M[MAR]: PC ← C; | MDR=M(4)=Ld, PC=8 |
| T10 | IR ← MDR; | IR=Ld |
| Ejecución | Ídem anterior – para vagos creativos | |
| T11 | A ← (rb = 0 → 0: rb ≠ 0 → R[rb]); | A=0 |
| T12 | C ← A + (16@IR⟨16⟩#IR⟨15..0⟩); | C=00000104H |
| T13 | MAR ← C; | MAR=00000104H |
| T14 | MDR ← M[MAR]; | MDR=M(104H)=3 |
| T15 | R[ra] ← MDR; | R(2)=3 |

# Ejecución: *Add r3, r2, r1*

- PC = 00000008H, r1=2, r2=3

| Paso | RTN para ld | |
|---|---|---|
| Fetch | Ídem anterior – para vagos creativos | |
| T16 | MAR ← PC: C ← PC + 4; | MAR=8, C=12=CH |
| T17 | MDR ← M[MAR]: PC ← C; | MDR=M(8)=Add, PC=12 |
| T18 | IR ← MDR; | IR=Add |
| Ejecución | | |
| T19 | A ← R[rb]; | A=R(2)=3 |
| T20 | C ← A + R[rc]; | C=A+R(1)=3+2=5 |
| T21 | R[ra] ← C; | R(3)=C=5 |

# Ejecución: *St r3, 0, 0108H*

- PC = 0000000CH, r1=2, r2=3, r3=5

| **Paso** | **RTN para ld** | |
|---|---|---|
| **Fetch** | **Ídem anterior – para vagos creativos** | |
| **T22** | **MAR ← PC: C ← PC + 4;** | **MAR=CH, C=10H=16** |
| **T23** | **MDR ← M[MAR]: PC ← C;** | **MDR=M(CH)=St, PC=16** |
| **T24** | **IR ← MDR;** | **IR=St** |
| **Ejecución** | | |
| **T25** | **A ← (rb = 0 → 0: rb ≠ 0 → R[rb]);** | **A=0** |
| **T26** | **C ← A + (16@IR⟨16⟩#IR⟨15..0⟩);** | **C=00000108H** |
| **T27** | **MAR ← C;** | **MAR=00000108H** |
| **T28** | **MDR ← R[ra];** | **MDR=R(3)=5** |
| **T29** | **M[MAR] ← MDR;** | **M(108H)=5** |

# Conclusiones del Ejemplo

- **El CPU es sólo una máquina.**
  - **Siempre hace lo mismo – ciclo de la instrucción.**
- **El Programa es quien da la "inteligencia".**
  - **Hay un programa que sintetiza un algoritmo.**
- **Al final el PC apunta a la siguiente instrucción.**
  - **Continúa indefinidamente.**
  - **"Para" con un retorno al Sistema Operativo.**
  - **Pero en el fondo no para, ejecuta el sistema operativo.**

# Performance del Ejemplo

- **Tiempo de Ejecución: 30 T**
- **Cantidad de Instrucciones: I = 4.**
- **Ciclos T por Instrucción Promedio - CPI.**
  - **Depende del programa.**
  - **En este caso resulta: 7,50.**
  - **Tiempo de ejecución = I x CPI x T**
  - **Aparecen las 3 variables que se pueden mejorar.**
    - **I: un buen ISA, buen programa, algoritmo, estruct. datos**
    - **CPI: mejorar la realización concreta del ISA.**
    - **T: mejorar la tecnología para usar un reloj más rápido.**

# Lenguaje de Máquina (LM)

- **Conformado por las Instrucciones del ISA.**
  - **Tal como son cargadas en memoria y leídas por el CPU.**
  - **Expresados como cadenas de ‘0’ y ‘1’.**
  - **Las primeras CPUs se programaban así.**
- **Problemas**
  - **Propenso a errores: basta con cambiar un 1 por un 0.**
  - **Ilegible.**
  - **Indocumentable.**
  - **Requiere mucho tiempo de las personas.**
  - **Al principio lo caro eran las computadoras.**
- **Escribir en Hexadecimal es un primer avance**.

# Lenguaje Assembler (LA)

- **Conformado por las Instrucciones del ISA.**
  - **Pero escritas en código mnemotécnico.**
  - **Correspondencia uno a uno con LM.**
  - **Es necesario traducirlas antes de ejecutarlas.**
- **Traductor = Ensamblador.**
  - **Tarea mecánica realizada por un programa traductor**
  - **Instrucciones al traductor = Seudoinstrucciones.**
  - **Incorpora también símbolos para variables y rótulos.**
  - **Ejemplos ORG, DS, DC, EQU...**
  - **Acepta Comentarios**

# Ej: programa en Lenguaje Ensamblador

**Algunas Seudoinstrucciones:**

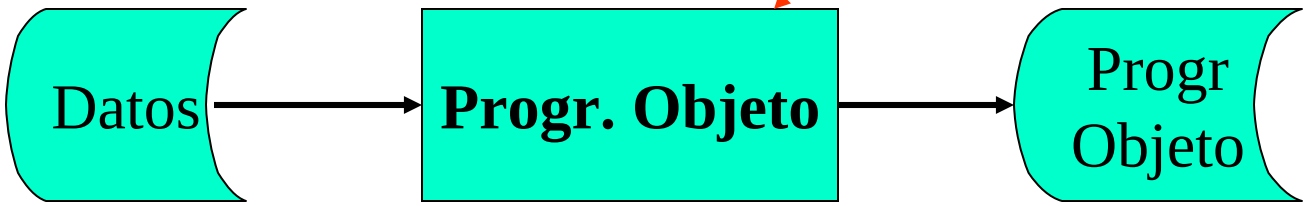
| | | |
|---|---|---|
| DCW | Define Constant Word | |
| DSW | Reserve Storage Word | |
| ORG | 100H | Ubicar el código siguiente a partir de 100H |

```
        org     000H        ;comienza en 000H
        ld      r1, 0, Dos  ;r1 = 2
        ld      r2, 0, Tres ;r2 = 3
        add     r3, r2, r1  ;r3 = r2 + r1 = 2 + 3 = 5
        st      r3, 0, Res  ;M(108H) = r3 = 5
fin:    br      fin         ;autolazo indefinido
        org     100H        ;los datos a partir de 100H
Dos:    dcw     2H
Tres:   dcw     3H
Res:    dsw                 ;reservar 1 palabra para result.
        end
```

# Ensamblador

**Traducción: de Fuente (LA) a Objeto (LM)**


**Ejecución: Corre el programa Objeto**



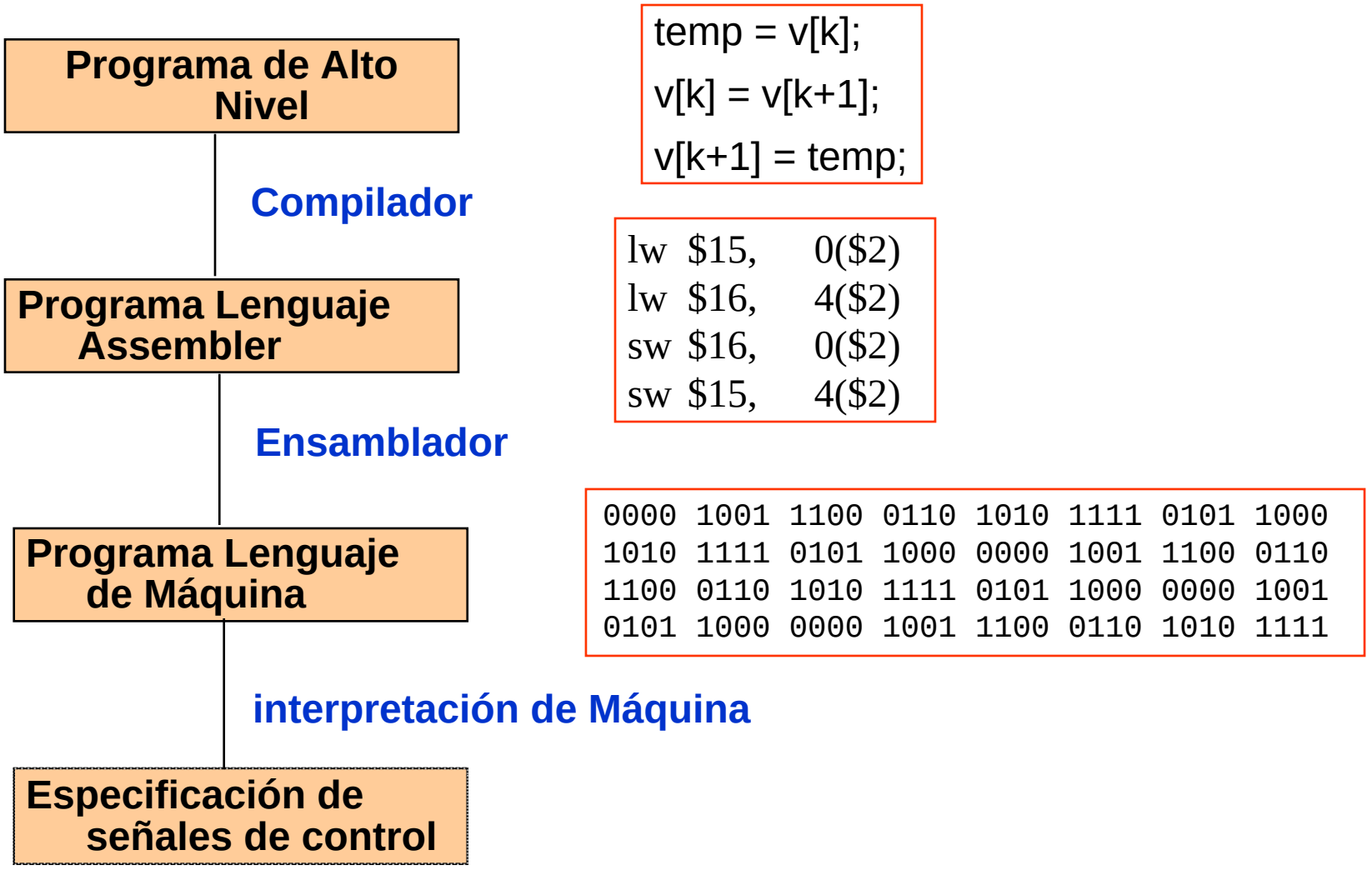
# Ventajas y Desventajas LA

- Desventajas
  - Cada máquina tiene un LA diferente
  - LA está orientado a la máquina y no al usuario.
  - No hay programación estructurada.
  - Hay que ser un experto.
  - Difícil desarrollar mercados...
  - ¿Y por qué no se ponen de acuerdo en un solo LA?
- Ventajas
  - Manejo exacto de Tiempo y Memoria.
  - Para aplicaciones críticas en tiempo real.

# Lenguaje Superior (LS)

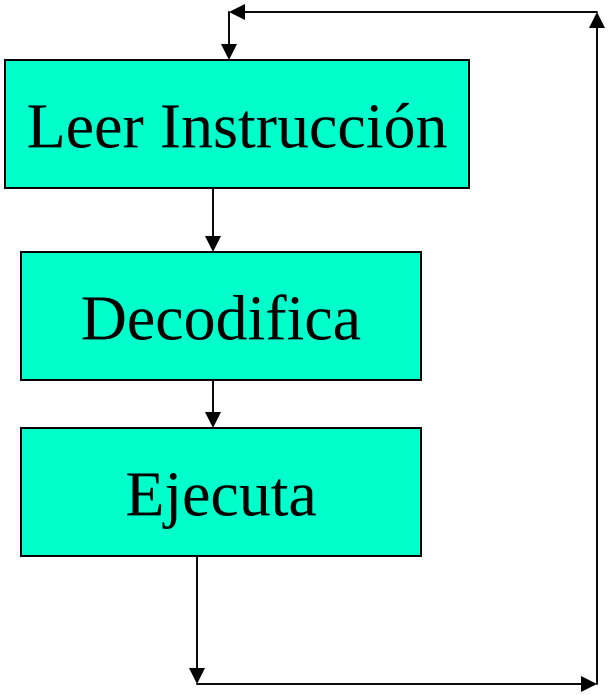
- Característica: 1 inst. LS → n inst. LM
- Orientado a Problemas y a Usuarios.
- Independiente de la Máquina.
- ¿Ejemplos de Lenguajes Superiores?
- Traductor = Compilador. Traduce a LM o LA
- Traducción no es única, optimiza.
- Primero se compila, luego se carga y ejecuta.

# Compilador, Ensamblador, Lenguaje de Máquina



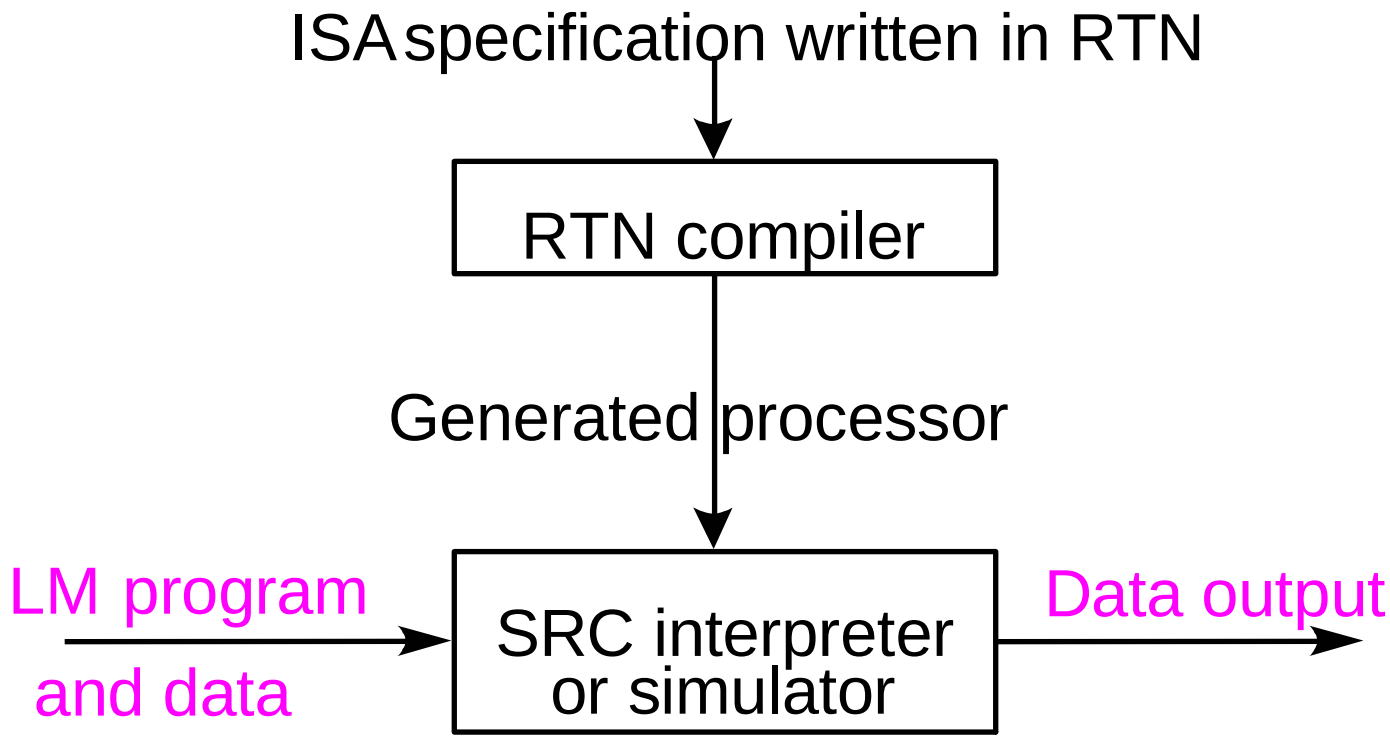
# Intérprete

- **Características similares al Compilador.**
- **Pero traduce y ejecuta en un solo paso.**
- **¿Cuándo conviene Compilador y cuándo Intérprete?**
- **Observación: el control del CPU es un intérprete para ISA.**



# Lenguajes de Especificación - RTN

**Lenguaje que describe otro lenguaje (LM) se llama Metalenguaje**



# Ventajas de RTN

- Permite describir el “qué” sin obligar a describir el “como”.
- Especificación precisa y no ambigua,
  - **al contrario de lenguajes naturales.**
- Reduce errores:
  - **Debido a mala interpretación o especificaciones imprecisas.**
  - **Confusiones en diseño e implementación —“error humano”**
- ¡Permite al diseñador hacer chequeo de la especificación!
- Las especificaciones pueden ser chequeadas y procesadas automáticamente mediante herramientas CAD.
  - **Una especificación RTN puede ingresar a un generador que**
    - **Produce un simulador para la máquina especificada.**
  - **Una especificación RTN puede ingresar a un generador de compilador.**
    - **Genera un compilador para el lenguaje de la máquina en cuestión.**
    - **La salida del compilador puede correr en el simulador.**

# Sistema Operativo

- ¿Cómo trabajar con un Computador?
  - ¿cómo cargo un programa?
  - ¿cómo manejo discos y monitor?
    - No basta con el Hw puro.
    - No basta con el ISA.
- Se necesitan Interfases.
    - Físicas (Hw – I/O: teclado, monitor, ...)
    - Lógicas (manejadores del Hw, Intérpretes).
- Y Optimizar uso de recursos...
    - Tanto de Hw como de Sw.
    - Compartir.
    - Presentar interfases transparentes a emplear por el usuario.
    - Seguridad, confiabilidad.

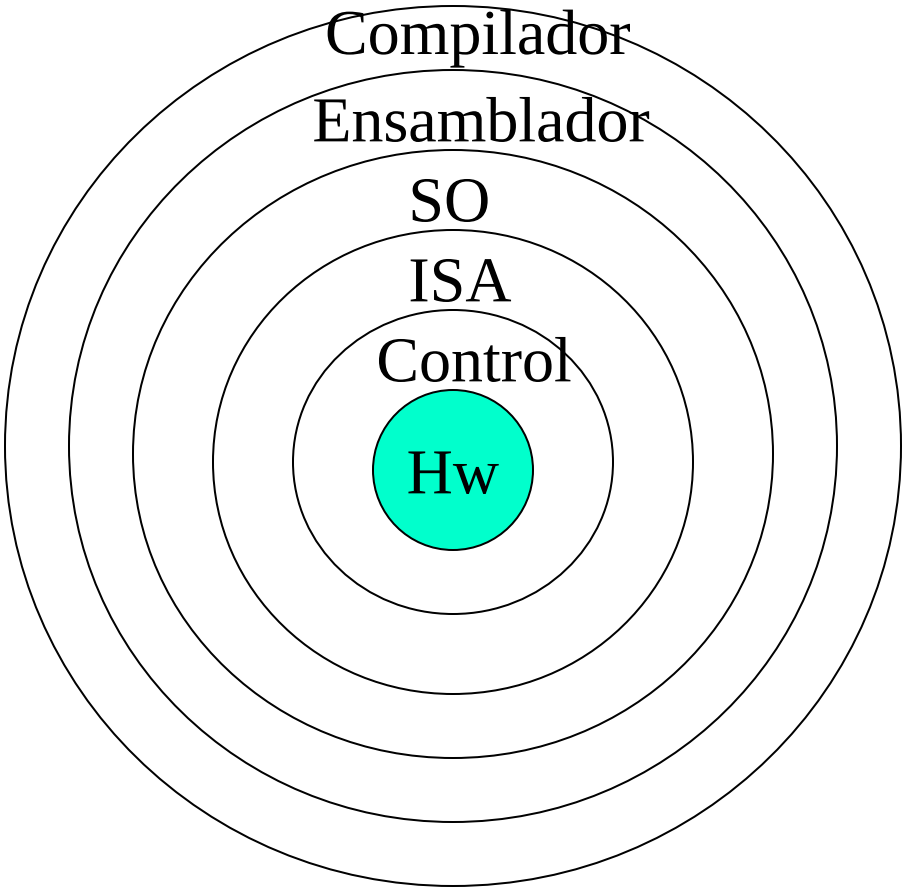
# Multiprogramación: Conceptos

- Recursos: CPU, Memoria, Disco, Printer, ...
- Acceso a Disco equivale a 100.000 ciclos de CPU.
- Si corre un único programa →CPU no hace nada mientras se accede a Disco. ¡Ineficiente!
- Pero se podría poner a otro programa para que use el CPU “concurrentemente”.
- Así, para la eficiencia de todos los recursos, conviene que corran varios programas concurrentemente.
- El Sistema Operativo distribuye los recursos de manera equitativa y transparente.
- Cada programa siente que corre solo.

# Multiprogramación – Conceptos

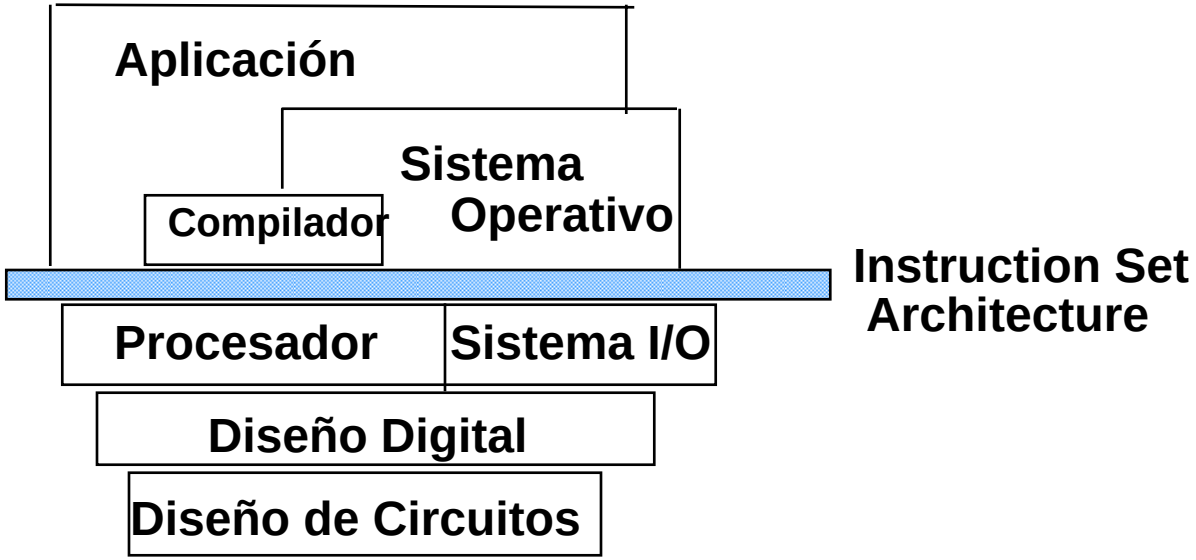
- **Ningún programa puede afectar a los demás.**
- **El Sistema Operativo administra el uso compartido de recursos y asegura que todo funcione bien (protecciones)**
- **La memoria es un recurso más. Un programa puede estar en disco en un momento dado.**
- **Es necesario que un programa funcione correctamente, independientemente de donde se cargue en memoria principal.**

# Visión Jerárquica por Niveles



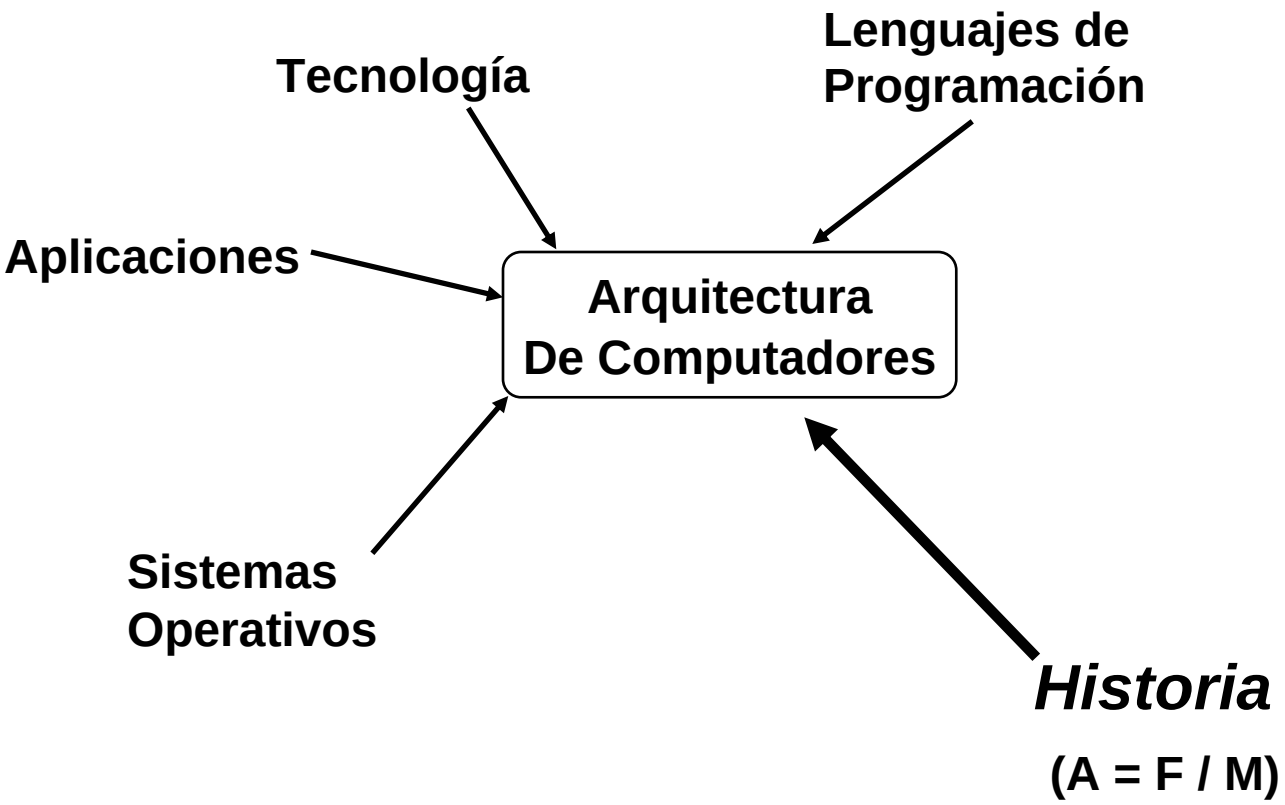
# Arquitectura de Computadoras

° Cordinación de ***niveles de abstracción***




° Bajo presión de fuerzas que cambian muy rapidamente

# Fuerzas en Arq. de Computadoras



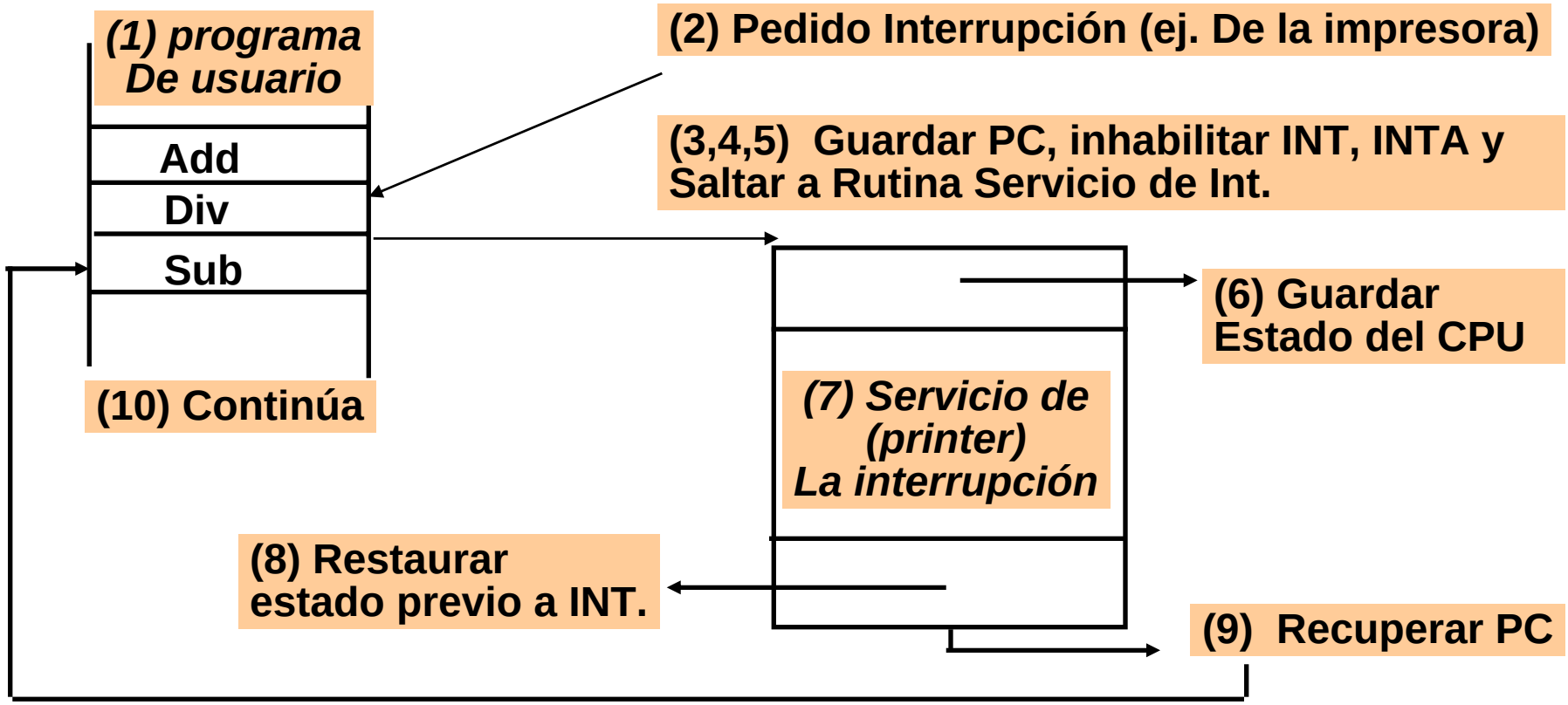
# Interrupciones en la Vida Real

1. **Un estudiante *ejecuta una tarea*: estudia.**
2. **Le golpean la puerta: le llaman por teléfono.**
   - ***Interrupt Request*.**
3. **¿Atendera? – *Consulta de Prioridades*.**
4. **Indica que sí – reconocimiento de la Interrupción**
5. **Pone un señalador en su libro.**
   - ***Guarda el estado de su tarea*.**
6. **Atiende el teléfono – *sirve a la interrupción*.**
7. ***Recupera el estado de su tarea*.**
8. **Continúa con la tarea original.**

# En un Procesador

1. **Un CPU ejecuta una tarea (corre un programa)**
2. **Impresora externa requiere nuevos datos a Imprimir – pide interrupción.**
3. **Prioridad(programa) < Prioridad (Interrup) → se atiende.**
4. **Señal de Salida – reconocimiento: INTA.**
5. **Se guarda PC (en algún registro o en Stack).**
6. **PC ← Comienzo Rutina Interrupción.**
   - **Se Ejecuta Rutina de Servicio Interrupción.**
     - **Al comienzo guarda toda información adicional sobre el Programa.**
7. **Al terminar recupera información y PC ← $PC_{ant}$**
8. **Continúa tarea inicial.**

# Punto de Vista del Programador



# Detalles de Implementación

- **Pedido de Interrupción**
  - **Una o más líneas Externas de entrada al CPU.**
- **Prioridad.**
  - **CPU lleva prioridad del programa en PSW (un valor).**
  - **Cada línea de interrupción tiene una prioridad asignada.**
- **Guardar Estado.**
  - **El PC se guarda en el stack o en otro registro.**
  - **El resto del estado se guarda en el stack.**
  - **Más adelante en el curso veremos por qué en stack.**
- **Rutina de Interrupción.**
  - **Se carga PC con la dirección de la Rutina de Interrupción.**
  - **Guardar viejo valor de PSW y poner nueva prioridad.**
  - **Se enmascaran automáticamente por Hw las Interrupciones. ¿Por qué?**

# Detalles de Implementación

- **Recuperar Estado.**
  - **La Rutina recupera de Stack parte por Sw.**
  - **Recuperar $PSW_{ant}$.**
  - **RTI o RTE: $PC \leftarrow PC_{ant}$.**
- **En General: Sistema Multitarea.**
  - **¿Puede un usuario modificar las prioridades?**
  - **¿Acceder a recursos compartidos?**
  - **El Sistema Operativo (SO) se encarga de servir Interrupciones.**
  - **PSW informa si ejecuta en modo SO o modo Usuario.**
  - **Instrucciones "privilegiadas" solo corren en modo SO.**
- **Característica de las Interrupciones: Asincrónicas.**
  - **Deben ser transparentes al Programa.**

# Múltiples Fuentes de Interrupción

**Suena el Teléfono, el timbre y llaman a almorzar...**

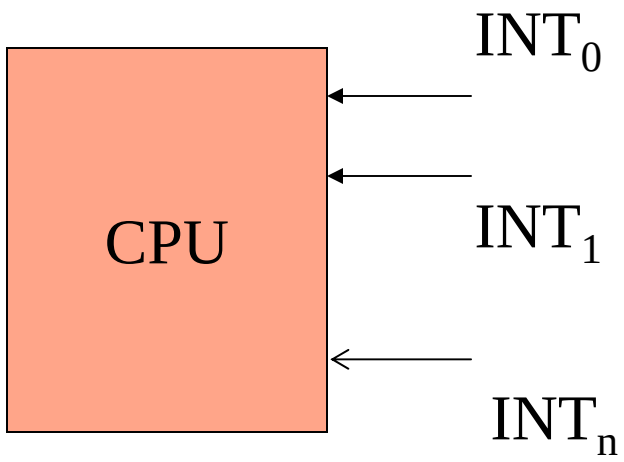
**Problemas**

1. **Conexión.**
   - **¿Cómo se conectan al CPU?**
2. **Identificación.**
   - **¿Cómo se identifica quién pide Interrupción?**
3. **Simultáneas**
   - **¿Qué pasa si se pide más de una a la vez?**
4. **Anidamiento**
   - **Mientras se sirve una Interrupción aparece otra.**

# 1.1 Conexión.

## Alternativas:

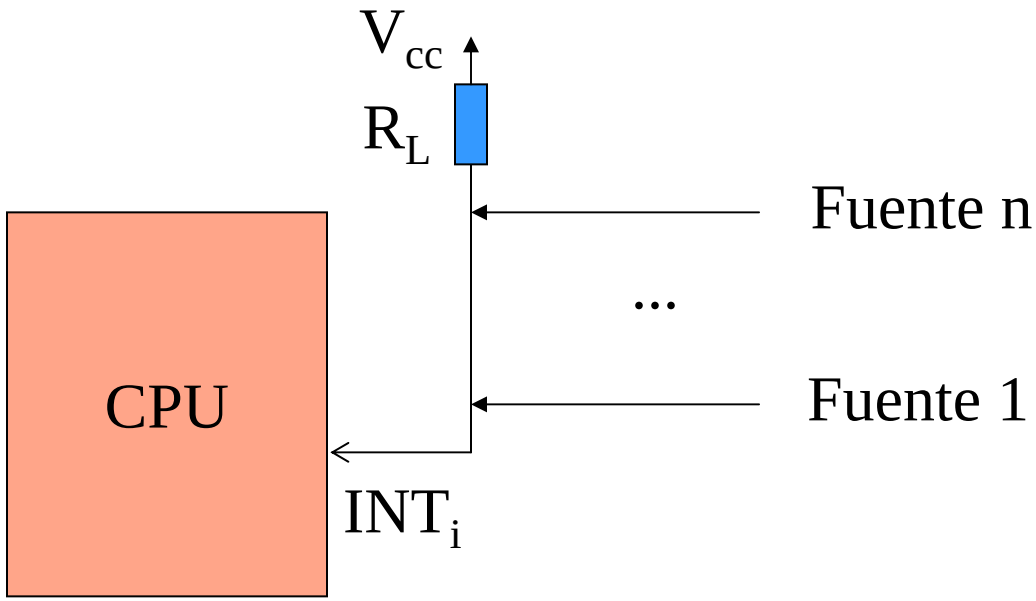
### 1. Múltiples Líneas de Pedido.



# 1.2 Conexión.

## Alternativas:

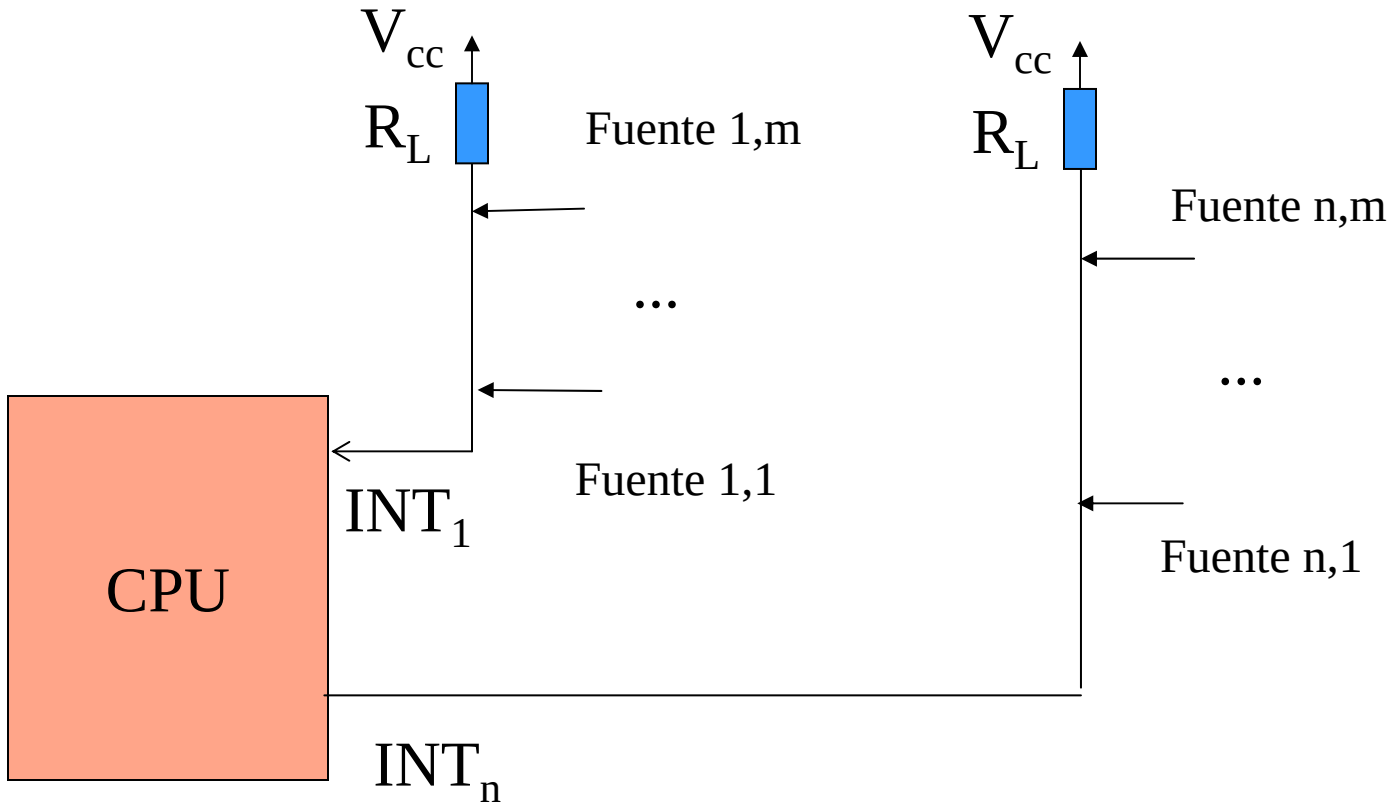
### 2. Una línea con conexión Open Collector.



# 1.3 Conexión.

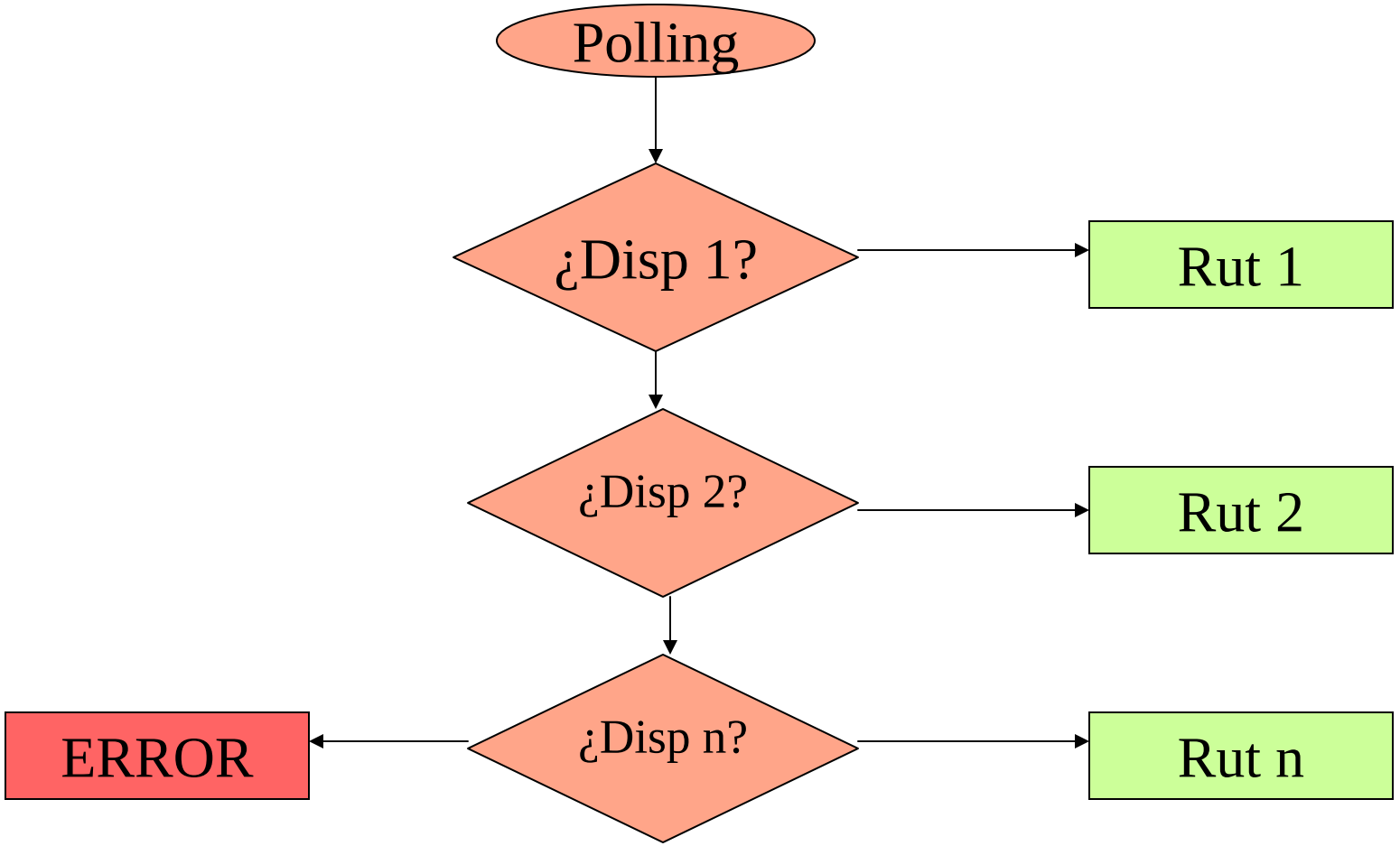
## Alternativas:

3. **Mixto (cuando hay más fuentes que líneas).**



# 2.1 Identificación por Sw.

**Analogía: Camarero de Hotel**



# 2.2 Identificación por Hw

1. **Múltiples Líneas.**
   a) **Una Dirección de Servicio para cada línea.**
   b) **Registro de Causa.**

   **Hw escribe automáticamente qué linea fue en Reg. Causa.**

   **Para identificar quién fue dentro de la línea:**
   - **Hw Externo – más adelante.**
   - **Polling – Sw.**

2. **Interrupciones Vectorizadas.**

   **En respuesta a INTA, el dispositivo se identifica.**

   **El CPU lee identificación (Vector de Interrupción, VI).**

   **VI apunta a un lugar de una tabla de direcciones de Rutinas de Servicio.**

   **Hw Salta automáticamente a la rutina de servicio adecuada.**

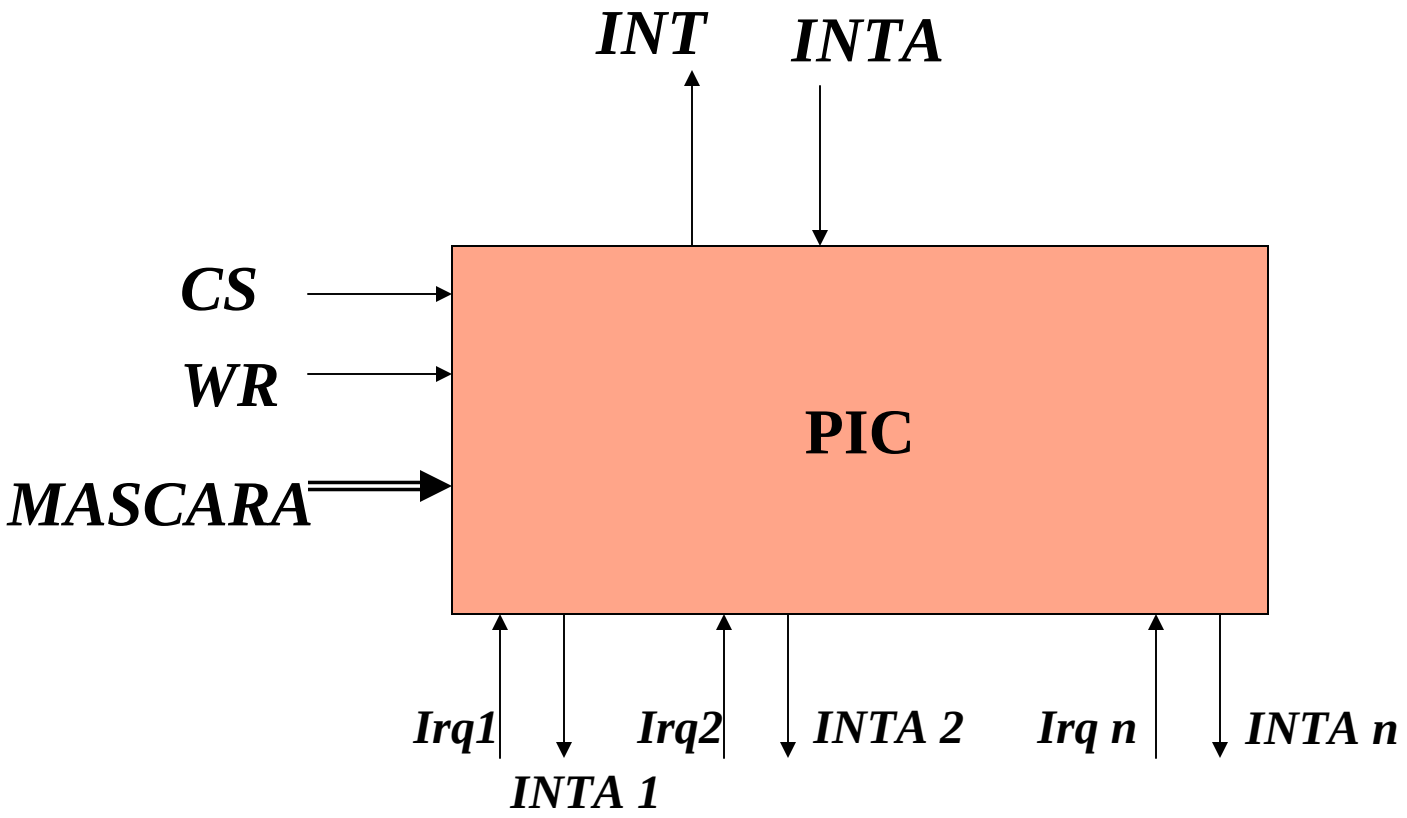
# ¿Ventajas y Desventajas?

- **Sw.**
  - **Ventajas:**
  - **Desventajas:**
- **Hw**
  - **Ventajas:**
  - **Desventajas**:

# 3. Simultáneas - Prioridades

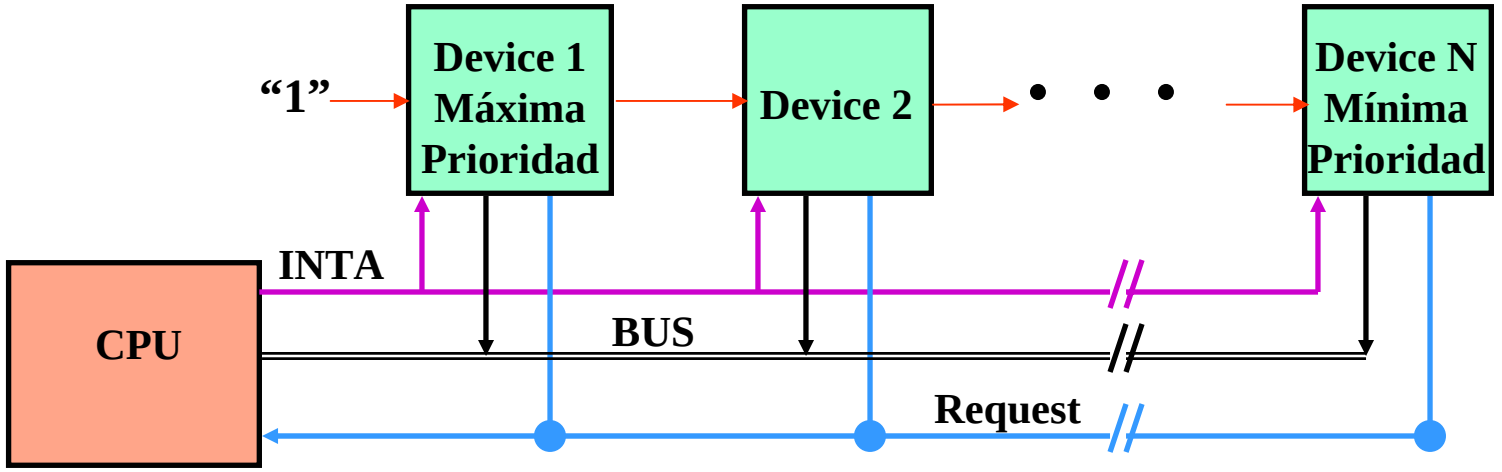
1. **SW - Polling.**
   - **El que se encuesta primero es prioritario.**
   - **Permite esquemas elaborados más equitativos.**
2. **Hw.**
   - **Cada línea tiene su prioridad.**
   - **Hw externo para resolver prioridades.**
     a) **PIC – Priority Interrupt Controller**
     b) **Conexión Daisy Chain.**

# PIC – Priority Interrupt Controller

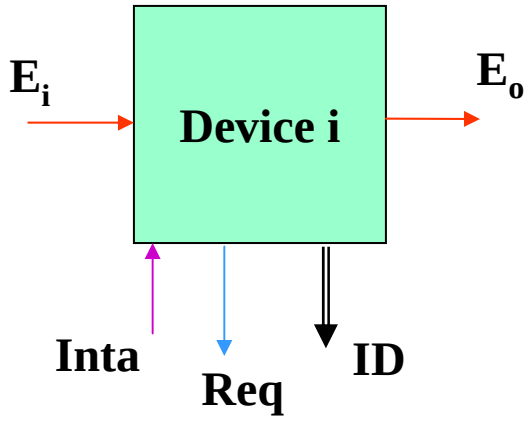



**Quien resulte elegido se identifica en el DATA BUS.**
**Modular: Se pueden armar árboles.**

# Conexión Daisy Chain




- **Funcionamiento:**
  - **Si $E_i$=0 →Req = 0, $E_o$=0**
  - **Si $E_i$=1 y no solicita Interrupción →$E_o$=1**
  - **Si $E_i$=1 y pide interrupción →Req=1, $E_o$=0**
- **Ventaja: simple y modular.**
- **Desventajas:**
  - **Retardo: admite un número máximo de dispositivos.**
  - **Si se rompe uno → falla para los siguientes.**



# 4. Anidamiento

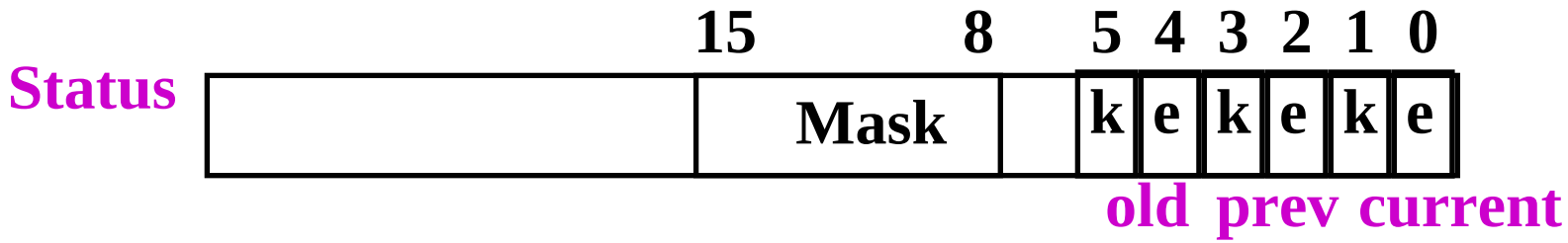
## Hablo por teléfono y suena el timbre...

1. **Define el Diseñador del SO.**
   - **Enmascaradas automáticamente por Hw.**
2. **Si Admite, entonces**
   i. **Guardar estado en stack y realizar actividades prioritarias.**
   ii. **Habilitar Interrupciones.**
3. **Por Sw es difícil → código reentrante.**
   i. **Debe ser código Puro.**
   ii. **Areas de datos independientes por cada entrada.**

# Excepciones e Interrupciones

- **Interrupciones**
  - **Causada por eventos externos**
  - **Asíncronas con la ejecución del programa.**
  - **Reclaman un servicio (por el Sistema Operativo)**
  - **Se ejecutan entre instrucciones del programa.**
  - **Simple: suspenda y continúe el programa del usuario.**
- **Traps o Excepciones**
  - **Causadas por eventos internos.**
    - **Condiciones excepcionales (overflow)**
    - **errores (parity)**
    - **Fallos (páginas de caché)**
  - **Sincrónicas con la ejecución del programa.**
  - **Se debe abortar la instrucción.**
  - **El problema debe ser remediado por el handler (rutina del SO).**

# Detalles del PSW en el MIPS




- Acepta 5 niveles de interrupción por Hw y 3 por software.
- Mask = 1 bit para cada nivel de interrupción.
  - 1 => habilita interrupciones del nivel correspondiente.
  - 0 => desabilita interrupciones.
- k = kernel/user (núcleo/usuario)
  - 0 => núcleo.
  - 1 => usuario.
- e = habilitación de interrupciones (de todas)
  - 0 => desabilitadas.
  - 1 => habilitadas.
- Cuando ocurre la interrupción, 6 LSB se corren 2 bits a la izquierda, poniendo los 2 LSB a 0
  - Corre en modo kernel con las interrupciones inhabilitadas.
- Volveremos sobre esto cuando veamos INTERRUPCIONES.

# Detalles del registro de Causa en MIPS


- Interrupciones Pendientes 5 niveles por Hw: se pone un bit en 1 si se solicita interrupción pero no se la responde todavía.
  - Sirve para manejar casos en los cuales hay más de un pedido de interrupción al mismo tiempo o cuando las interrupciones están inhabilitadas.
- Código de Excepción codifica la causa de la interrupción.
  - 0 (INT) => external interrupt
  - 4 (ADDRL) => address error exception (load or instr fetch)
  - 5 (ADDRS) => address error exception (store)
  - 6 (IBUS) => bus error on instruction fetch
  - 7 (DBUS) => bus error on data fetch
  - 8 (Syscall) => Syscall exception
  - 9 (BKPT) => Breakpoint exception
  - 10 (RI) => Reserved Instruction exception
  - 12 (OVF) => Arithmetic overflow exception

# En perspectiva: Modos Supervisor / Usuario

- **Mediante los dos modos de ejecución (user/system) es posible que el computador se auto-administre adecuadamente.**
  - **El Sistema Operativo es un programa especial que corre en modo privilegiado y tiene acceso a todos los recursos del computador.**
  - **presenta "recursos virtuales" para cada usuario que son más convenientes que los recursos físicos.**
    - **archivos en lugar de sectores de disco.**
    - **Memoria Virtual en lugar de Memoria Física.**
  - **protege a cada programa de usuario de los otros.**
- **Las Excepciones le permiten al Sistema Operativo tomar acción en respuesta a eventos que ocurren cuando el programa de usuario está corriendo.**
  - **El SO comienza a partir del manejador (Handler).**
  - **El programa usuario solicita servicios mediante pedidos de interrupción por Sw (ej. I/O).**

# Modo Trazador

- **Algunos CPU tienen un flag en PSW, T.**
- **Si T=1:**
  - **Excepción al finalizar cada Instrucción.**
  - **Pasa a modo Supervisor y automáticamente T=0.**
  - **Servicio de la Excepción (debugging).**
  - **Al retornar, T=1**
- **¿Por qué T=0 al responder la Interrupción?**

# Función de RESET

- **Asegura que la máquina comience desde un estado cierto.**
  - **El PC se fija en un valor determinado.**
    - **Un valor fijo por Hw.**
    - **El contenido de memoria (ROM) cuya dirección es fija por Hw.**
  - **PC apunta a la Rutina de Inicialización o RESET (ROM)**
  - **Rutina de Inicialización realiza booteo del SO.**
  - **Las interrupciones quedan enmascaradas.**
  - **Modo Supervisor.**
  - **Estado inicial debe asegurar el Inicio (T=0, por ej.).**
  - **Warm Reset Vs Cold Reset**
    - **WARM, por SW, llama a rut Inicialización.**

# Ejemplo: Excepciones en M68000

- (1) Cambio PSW
  - Se realiza una copia temporaria del viejo PSW.
  - Bit S=1 y T=0 en PSW (modo supervisor, sin trazador)
- (2) Obtención de la dirección del Vector de Excepción (VE).
  - Se lee ID de 8 bits y se le agrega 00 a la derecha.
  - El contenido de la dirección dada por ID.00 es VE.
    - Hay una tabla de 256 entradas con los VE
  - En VE comienza el manejador de la excepción (handler)
- (3) Los viejos PC y PSW se guardan en el stack del sistema.
- (4) Se carga PC con VE.
- El retorno del handler se hace con RTE.
  - Recupera PSW y PC.

# Manejo de Prioridades

- Línea única (INT) para pedir Interrupción.
- 3 líneas para informar prioridad.
- Se permiten 7 niveles de prioridades.
  - La prioridad 000 no provocaría nunca una interrupción.
- PSW contiene 3 bits que indican prioridad actual.
- Se ignoran las excepciones cuya prioridad sea menor o igual que la actual.

# ¿Qué Hemos Visto?

- Computadora 3 partes: CPU, M, I/O.
- I/O – gran diversidad de dispositivos.
  - Eficiencia (cafetera):
    - Polling.
    - Interrupciones.
      - Uno o más fuentes de interrupciones.
      - Anidadas (como recursivos) o no.
      - Conexión por Sw o Hw.
      - PIC, Daisy Chain.
    - DMA.
    - IOP.
    - Ayuda del Sistema Operativo – Transparencia y Seguridad.

# Sistema de Memoria

- Clasificación.
  - Tecnología: Dinámica, Estática…
  - Modo de Acceso: RAM, Secuencial.
  - Acceso del CPU: Primario o Secundario.
- Cache y Memoria Virtual.
  - Principio de Localidad Espacial y Temporal.
- Conexión al CPU – MAR, MDR.

# CPU

- Ciclo de la Instrucción.
- Camino de Datos.
  - Posibilita físicamente que se ejecuten las Inst.
  - Registros Especiales: PC, IR, MAR, MDR, PSW, CAUSA.
- Control.
  - Señales para que se hagan las cosas.
- Ejemplo SRC-1. Laboratorio. RTL.

# Sistema de Varios Niveles.

- Hw.
- Control.
- ISA.
- Sistema Operativo.
- Ensamblador.
- Compilador, Intérprete.
- ORGANIZADO EN CAPAS.

# Sistema Operativo

- Multitarea / Multiusuario.
  - Soporte de Arquitectura: Usuario / Supervisor.
  - Interrupciones.
- Manejo de Recursos.
  - Transparencia y Facilidad de Uso.
  - Equitatividad.
  - Eficiencia.

# En Síntesis

- Sistema Complejo.
- Sometido a grandes fuerzas de Cambio y de Inercia.
  - Tecnología →
  - Nuevos Lenguajes →
  - Sistemas Operativos →
  - Nuevas Aplicaciones. →
  - Aplicaciones existentes (Historia) ←