

RESUMEN FINAL

ORGANIZACIÓN DEL COMPUTADOR

CÁTEDRA: BENÍTEZ

ALUMNA: MARIA PAULA BRUCK

Resumen final

U1 → Sistemas numéricos

* Ceros especiales de cambio de base →

$$b^x = p$$

base inicial

Enteros:

* A base decimal → $ABCD_b = (A \cdot b^3 + B \cdot b^2 + C \cdot b^1 + D \cdot b^0)_{10}$

* desde base decimal → divisiones sucesivas → hasta q el cociente < al

* de b a p → ① de b a decimal

② de decimal a p.

↓ divisor

punto ultimo cociente
↓ todos los restos de abajo
hacia arriba.

Fracciones:

* A base decimal → $0.ABC_b = (A \cdot b^{-1} + B \cdot b^{-2} + C \cdot b^{-3})_{10}$

* desde base decimal → multiplicaciones sucesivas → se toma de cada resultado
↓ termina cuando le pone la parte entera y la parte
fraccionaria es igual a 0 / fraccionada es la que se le
suman + decimales se tienen multiplicar a la base que se
+ precisan.

quieres obtener.

* de b a p → ① de b a decimal

② de decimal a p.

Métodos de representación de números negativos.

* Signo y magnitud → 0 → si el nº a representar → +

↓ min: -2^{n-1} → 1 →

→ 0

cont valores → el signo → el resto para el módulo del nº a representar en
el 0 → el signo → el resto para el módulo del nº a representar en
absolutos → el 0 → el signo → el resto para el módulo del nº a representar en
máx: $2^{n-1} - 1$ → el signo → el resto para el módulo del nº a representar en

→ Varios → rango simétrico → pueden representarse misma cantidad de nº → - q +

Diferencia → doble representación → los 2 valores absolutos de los puntos son el mismo

→ No permite operar → aparece el 0 signado: (+0,10) y (-0,10) → excepto 1 lugar
entre medias → q no tiene signo.

→ No puedo hacer cuentas con valores q representan 0 q no tiene signo.
números reales

* Complemento a 1 → 0 → si el nº a representar $\rightarrow \oplus$

min: -2^{n-1}
max: $2^{n-1} - 1$

cont valores → 1 → si pose el signo el resto para el modelo del nº a representar en binario. Si es negativo se le aplica el NOT

Ventajas → nro simétrico → pueden representarse misma cantidad de nº \ominus q \oplus
 permite operar aritméticas → los 2 valores absolutos de los puntos son el mismo para poder obtener el resultado correcto se debe sumar el acercamiento obtenido al final de la suma/resta

Deseventaja → doble representación del 0 → aparece el 0 signado: $(+0_{10})$ y (-0_{10}) → excepto 1 lugar q no necesita el 0 no tiene signo

* Complemento a la base → Rep $(x_b) = x_b$ si $x \geq 0$

Rep $(x_b) = C_b(1|x_b|)$ si $x < 0$
 $C_b(r) = B^n - r$

min: $-(B^n/2)$
 max: $(B^n/2) - 1$

Ventajas → única representación del 0
 Deseventajas → permite operar aritméticas → nro simétrico → puede representarse un \ominus mas.

* Complemento a 2 → Rep $(x_{10}) = x_2$ si $x \geq 0$

Rep $(x_{10}) = \text{NOT}(1|x_2|) + 1$ si $x < 0$

min: -2^{n-1}
 max: $2^{n-1} - 1$

Ventajas → única representación del 0
 Deseventajas → permite operar aritméticas → nro simétrico → puede representarse un \ominus mas.

* Exceso a base → Rep $(x_b) = x_b + \text{exceso}$

Deseventajas → requiere de operaciones $B^{n/2}$ intermedios, si se combinan el nº de bits se combina el exceso
 nro simétrico → puede representarse un \ominus mas.
 Ventajas → Permite operar aritméticas (q con el exceso)
 No hay encapsulación del nº → se formado + n-bits

representación en una determinada base de un número en un formato

Formato y configuración.

representación computacional de un número

* Expansión y truncamiento

complemento representación decimal digitos
un dígito el nº decimal dígitos
un dígito el nº

* Binario de punto fijo sin signo → quedar enteros \oplus

- ↳ ① Pasa n° a base 2
- ↳ max: $(2^n - 1)_{10}$ / min: 0
- ② Completo 0 a izq para llegar a la base 2

* Binario de punto fijo con signo → quedar enteros \oplus y \ominus

- ↳ ① Pasa n° a base 2
- ↳ base 2
- ② Completo 0 a izq para llegar a la base 2
- ③ Si ($n^{\circ} \ominus$) complemento binario
complemento a 2 \rightarrow NOT + 1
- ↳ 1º bit se reserva para el signo \rightarrow 1 \ominus
- ↳ max: $2^{n-1} - 1$ / min: -2^{n-1}

Ventaja → validación overflow en operaciones aritméticas

revisa la capacidad } válida → últimos 2 dígitos iguales
de almacenamiento del formato de representación } inválida → ≠

* Empaquetado. → quedar enteros \oplus y \ominus

- ↳ ① Pasa a base 10
- ↳ base 16
- ② Coloco los dígitos en los nibbles
- ↳ max: $10^{2n-1} - 1$ / min: $-10^{2n-1} + 1$
- ③ Colocar en el último nibble el signo

CAFE $\overset{\oplus}{\rightarrow}$ BD

* Zonado → quedar enteros \oplus y \ominus

- ↳ ① Pasa a base 10
- ↳ base 16
- ② Coloco el dígito decimal en un nibble derecho
- ↳ max: $10^n - 1$ / min: $-10^n + 1$

③ Completar los nibbles ing. con F
excepto el último que indica el signo

CAFE $\overset{\oplus}{\rightarrow}$ BD

- * Binario punto flotante IEEE 754
- * Precision simple \rightarrow  $\rightarrow \text{exceso} = 127$
- * Precision doble \rightarrow  $\rightarrow \text{exceso} = 1023$
- ① me fijo el signo $\rightarrow \Theta = 1 | \Theta = 0$
 ② Poner a cero
 ③ Normalizar \rightarrow comienza con 1, ... \rightarrow nº de desplazamiento = exponente
 ④ Expo en exceso = exponente + exceso Derecha Θ Izquierdo Θ
 ⑤ puntos bit signo + expo en exceso + mantisa

02 → Maquina elemental

* Arquitectura de Von Neumann. → mecanización del tratamiento digital de la

Programa almacenado → información

↓ los instrucciones y los datos → flujo de ejecución

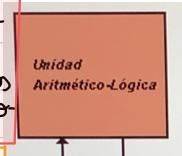
que en ellos se guardan residen
en una misma memoria

operaciones de decisión lógica automá-
ticos / existe una instrucción que permite
a la máquina no seguir con la
ejecución de ejecución.

↓ Esquema funcional

* realiza operaciones

aritméticos (+ -) y lógicos
(comparaciones)



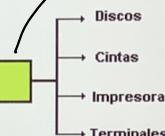
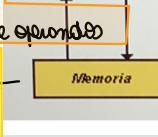
ESQUEMA FUNCIONAL
DE VON NEUMANN

* dirige/dirige todo lo
necesario para q una instrucción
de máquina se ejecute.



* hace instrucción, ordena, ejecuta operaciones

* dividida en celdas
con contenido variable.



* ayuda a transmitir
información.

* Se identifican x la dirección de memoria

* Capacidad total = cantidad de celdas → almacenamiento
instrucción del programa
datos = operando

* Registro → "memoria muy rápida" → permite almacenar cierta cantidad de

* compuerta → circuitos electrónicos bistables, bits en forma temporal
y unidireccionales → info en un solo sentido

televisor/mouse

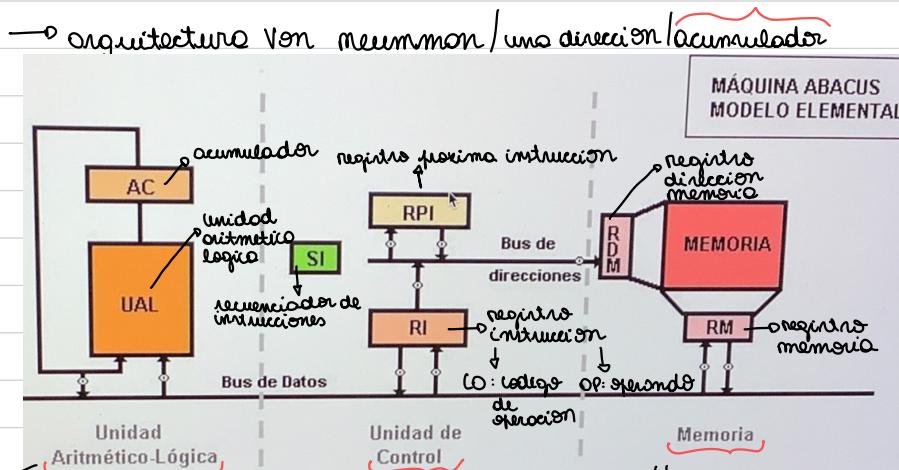
* Bus → de datos → mueve la info x los componentes del hardware

de control → de direcciones → ubica los datos en la memoria

Lo marca el estado de una instrucción q fue dada a la PC.

* Para el intercambio de info entre 2 registros se usa el bus y la compuerta debe estar abierta. Al cerrar la compuerta la info se pierde. No pueden existir 2 componentes abiertos

* Abacus



* acumulador → albergar 1^{er} operando
y resultado → int 1 res dirección
operaciones → largas (contenido de AC)
lógicos (OR, AND, XOE)
almacenamiento (enviar dato a memoria x el bus)
suma (sumar un dato a lo q haya en el AC)
todo dentro el acumulador

* enviar y analizar instrucciones
+ RPI → dar prox int a realizar
+ RI → instr extraída de memoria
+ CO → int res resultado de leer/escrito dentro en la memoria

administrar operandos → Si → distribuye los y cierra de componentes ordenes de UC a UAL y memoria para ejecutar los pasos de la int

* Tam → RPI = RDM = OP = cont celdas direccionables

→ AC = RI = RM = long int = long celda

- ④ UC ordena la transferencia del contenido del RPI al RDM y envía a la memoria una orden de lectura.
 ② El contenido de los celos de memoria quedo almacenado en el RM. La UC ordenó la transferencia del contenido del RM al RI
 ③ RPI se prepara para la próxima instrucción

* Fases → Busqueda → * localizar instrucción a ejecutar
 ↓ ejecución → * ejecuta la instrucción
 ↓ movimiento de los datos secuenciales y
 ordenadas → respuestas

* Suma → contenido RM + contenido AC
 $RDm \leftarrow (OP)$
 $RM \leftarrow ((RDM))$
 $AC \leftarrow AC + (RM)$

* Carga → almacena en el AC un dato contenido en memoria
 $RDM \leftarrow (OP)$
 $RI \leftarrow ((RDM))$
 $AC \leftarrow (RM)$

* Almacenamiento → guardar en memoria lo del AC
 $RDM \leftarrow (OP)$
 $RM \leftarrow (AC)$
 $(RDM) \leftarrow (RM)$

* Bifurcación → se debe saltar a la dir indicada en la instrucción. La dir de bifurcación debe ser transferida al RPI
 $RPI \leftarrow (OP)$

* Superabacus → cálculos se realizan en un solo sumador
 UAL calcula directo / con punto de registros generales (datos/direcc)
 sin datos
 des direcciones → No tiene RPI, se usa RO

Consulta: es la =

Suma inmediata

Superabacus

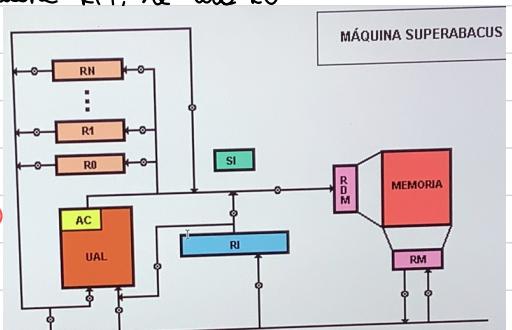
$AC \leftarrow (R1) + 100$

$R2 \leftarrow (AC)$

$AC \leftarrow (R2)$

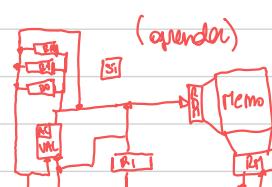
$R2 \leftarrow (AC) + 100$

g ento



modos de direccionamiento

- Inmediatos SUMAR R4 100
- Registro directo SUMAR R4 R5
- Registro Indirecto SUMAR R4 R5
- Base + Desplazamiento SUMAR R4 R5 100



* R1 → 2 operandos

$R1 \leftarrow$

$R11 - D$

codificación de operación

1 operando 2 operandos

* Fases → Busqueda → incremento del RO usando la UAL en vez del SI

↓ ejecución → suma registro → suma el contenido de otros registros/resultado en memoria → RM ← ((RDM))
 $AC \leftarrow (R2) + 100$
 $AC \leftarrow (R2) + (AC)$
 $AC \leftarrow (R2) + (AC) + 100$
 $AC \leftarrow (R2) + (AC) + (AC) + 100$
 $AC \leftarrow (R2) + (AC) + (AC) + (AC) + 100$

 Múltor lee el contenido de memoria → suma al registro en el 1 operando el contenido de la celda de memoria cuya dirección este dada + el contenido del registro indicado en el 2 operando + el offset / resultado en el 1 operando

$$AC \leftarrow (R3) + DO$$

$$R0M \leftarrow (AC)$$

$$RM \leftarrow ((R0M))$$

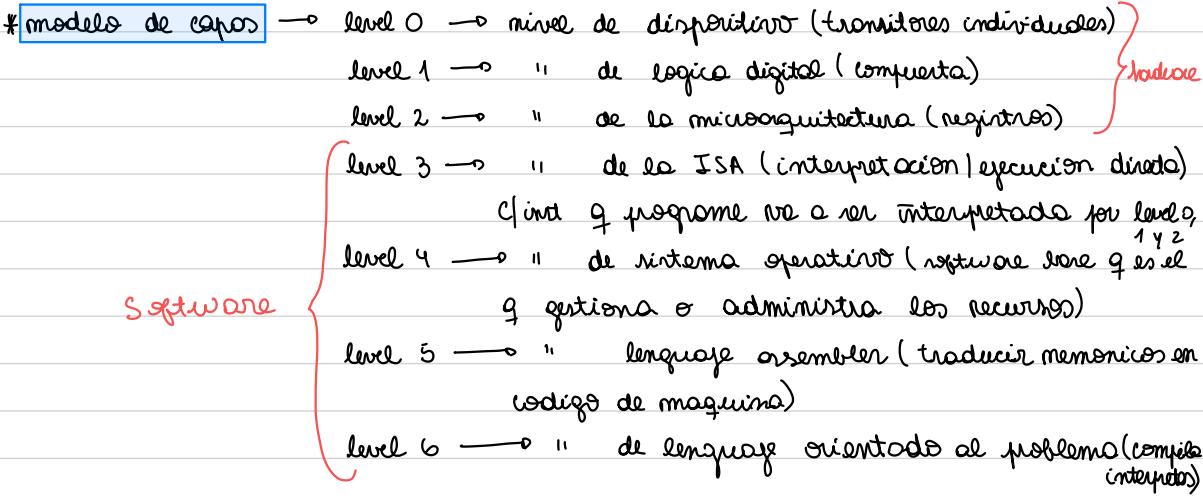
$$AC \leftarrow (RS) + (RM)$$

$$RS \leftarrow (AC)$$

* Cada instrucción Múltor tiene un código de operación ≠

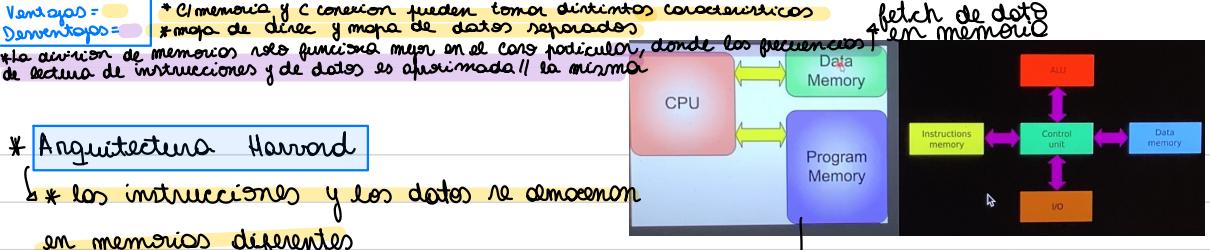
U3 → Arquitectura del conjunto de instrucciones

- * **Arquitectura de computadoras** → características computacionales visibles al programador
 - ↳ atributos q tienen impacto directo en la ejecución lógica de un programa
- * **ISA** → instrucción net architecture → especificación q el fabricante nos da de todos los atributos de la arquitectura del computador
 - ↳ * Repertorio de instrucciones: parte de la ISA, al programar tenemos saber cuál es el conjunto de instrucciones
 - * Especificación de su operación y como funcionan c/u / contratos
 - * Requerimientos: como se montan, mon en esa computadora
 - * Tipos de datos: tipos de aritmética, cant de bits
 - * modos de direccionamiento: formas de acceder a los operandos
 - * Formato de instrucciones: de q manera enton despliegan los campos dentro de una instrucción
 - * Memoria → word size (vinculado al tam del bus de datos) capacidad aritmética espec de ↳ Big/little endian
 - direcciones direcciónamiento
- * **Organización de computadoras** → como se implementa la arquitectura (microarquitectura)
 - ↳ mundo del electrónico / ms del informática
 - ↳ como los fabricantes implementan la lógica
 - ↳ implementaciones de una misma
 - ↳ consumo de energía
 - ↳ costos
 - ↳ velocidad de procesamiento
- microarquitectura → cereza (AR) ↳ bloques con impo q se reutilizan
 - ↳ microprogramación
- * **Familia de computadoras** → misma arquitectura base, ≠ organizaciones
 - ↳ modelos con precios y prestaciones ≠ pero compatibles entre si



* Clasificación de computadoras según su poder de cálculo

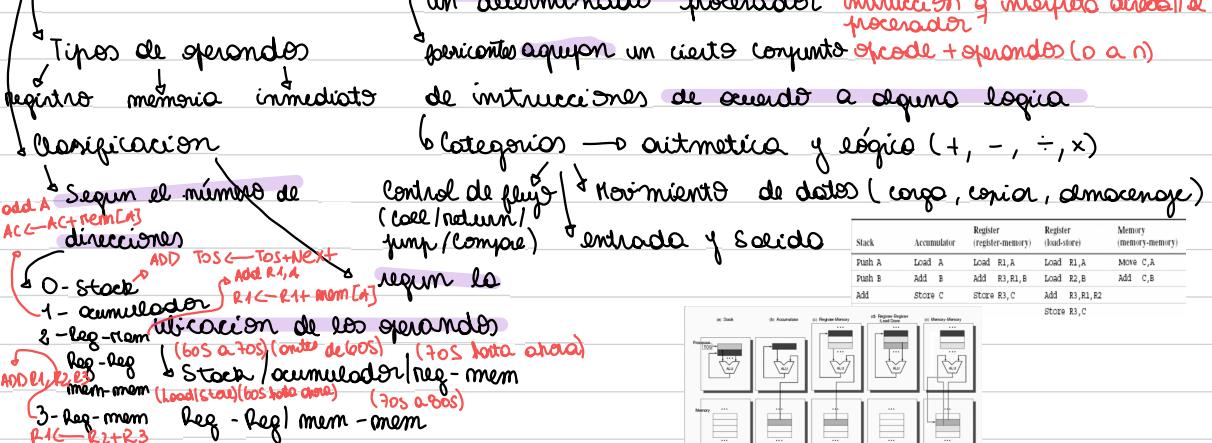
- Supercomputadores ↗ uso específico / uso comerciales
 - extremadamente rápidas
 - aplicaciones científicas / simulaciones / computo militar
 - manejan volúmenes de datos enormes miles de CPU
- Macocomputadores / Mainframes ↗ muy rápidos
 - banca / uso comercial / científicos / telecomunicaciones
 - manejan volúmenes de datos muy grandes
 - revendedores principales q puede tener una empresa
 - instituciones gubernamentales
- Minicomputadores / revendedores middle range ↗ rápidos
 - empresas media - uso comercial
 - manejan volúmenes de datos grandes
 - decenas de CPU
- Microcomputadores / PC → uso individual o redes pequeñas a medianas
 - uso hogareño / educativo / comercial
 - manejan volúmenes de datos no muy grandes
 - uno o varios CPU
- Computadores Portátiles / notebooks → uso individual portátil
 - uso hogareño / educativo / comercial
 - manejan volúmenes de datos no muy grandes
 - uno o varios CPU
- Computadora de mano → uso individual portátil acotado
 - uso hogareño / educativo / comercial
 - manejan volúmenes de datos pequeños
 - poseen 1 o varios CPU



* Arquitectura Harvard

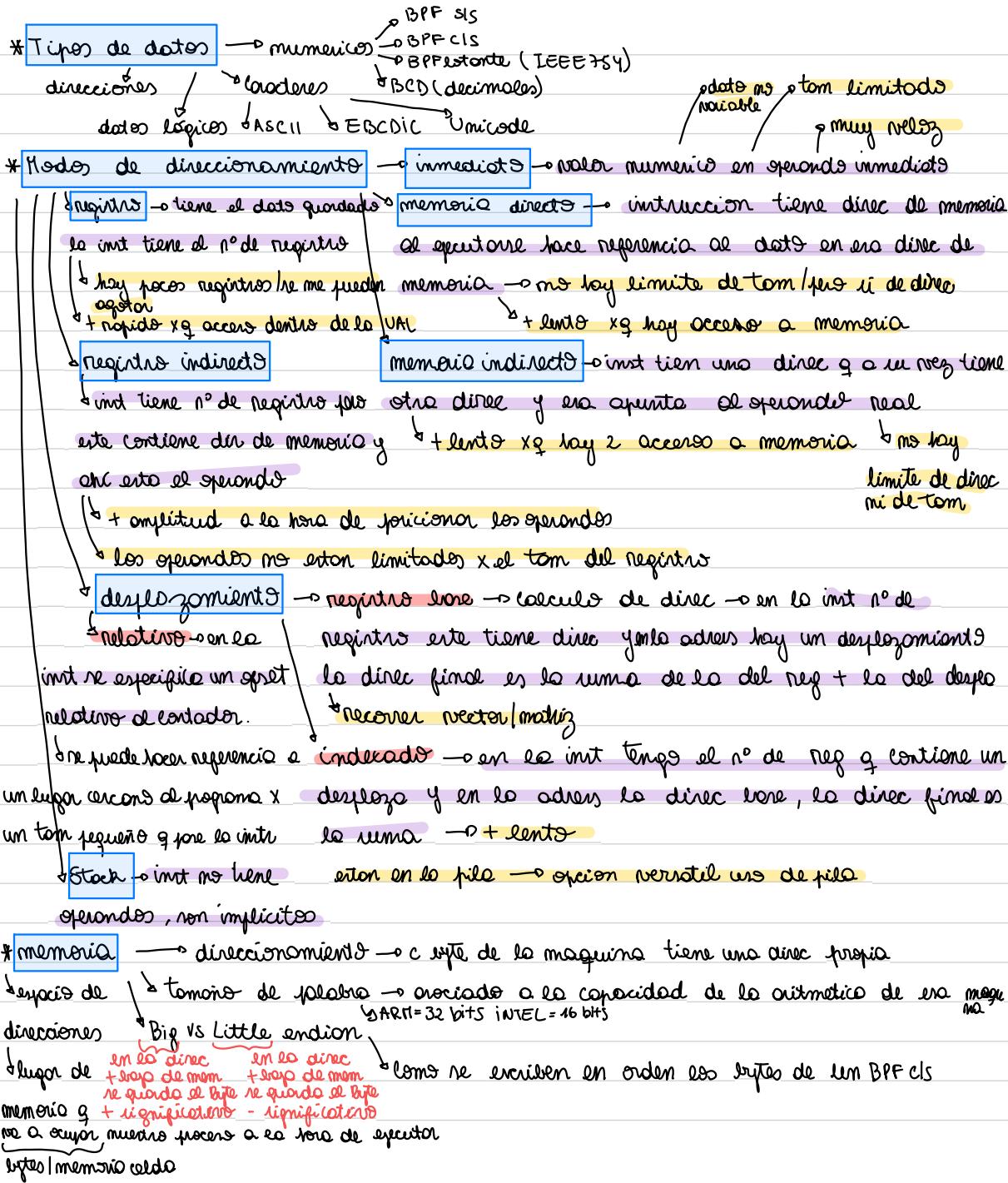
- * los instrucciones y los datos se almacenan en memorias diferentes
- * Hay 2 conexiones entre la unidad de control de la CPU y cada sistema de memoria
- * lo cual permite q se puedan cargar de mismo tiempo los datos e instrucciones
- * Al existir 2 tipos de memoria = se dificulta la programación porque hay q manejar espacios de direcciones =
- * Para usos específicos es viable → microcontroladores PIC / procesadores de señales digitales (DSP) para streaming de datos → > oncho de banda de memoria
 - ↳ oncho de banda + predecible

* Repertorio de instrucciones



* Formato de instrucciones (encoding)

- Componentes → opcode → define el despliegue de los bits q componen la instrucción
- flags → 0 o n operandos → modo de direccionamiento de cada operando
- Tipo → fijo → todos los int de máquina miden lo mismo (RISC)
- Variable → depende de como yo escriba esa instrucción no a terminar definida
- Múltiples de operandos → ≠ formularios + tam



- * Control de flujos
 - Compara cond branch
 - ↳ el if y el branch
 - enton contenidos en una misma invit
 - como resuelve la maquina una evaluacion logica y una validacion con condicional de acuerdo a esa validacion
 - Condition code → bits en memoria q eston retenidos y de acuerdo a como esten si bifurca o no
 - Condition register → 1 de los registros de la maquina q se cargo x el resultado de esa evaluacion logica, despues se cheque el reg

14 → Lenguaje ensamblador

- * **Lenguaje de máquina** → Es una representación binaria de un programa de secuencia de instrucciones computadora el cual es leído e interpretado por el de máquina
- * **Lenguaje ensamblador** → Representación simbólica del lenguaje de máquina de un procesador específico
 - ↳ Elementos q lo componen
 - ↳ Etiquetas → nombre q
 - ↳ Códigos de operación
 - ↳ Operando
 - ↳ Comentarios
 - ↳ Tipo de sentencias
 - ↳ Instrucciones
 - ↳ Directivas / pseudoinstr.
 - ↳ Comandos q le quiero transmitir al programa
 - ↳ Macroinstrucciones
 - ↳ Long de init / direct q al programar queremos q se repitan
 - ↳ ¿Por qué se usa?
 - Unve para referenciar una
 - Debugging / verificación
 - Desarrollar funciones óptimas de compiladores
 - Sistemas embutidos → Recursos limitados (memoria)
 - Drivers de hardware y código de sistema
 - Acceder a instrucciones no disponibles en un lenguaje de alto nivel
 - Código automodificable
 - Optimizar código en tamaño
 - Optimizar código en velocidad
 - Biblioteca de funciones
 - ↳ traduce mi programa a código de máquina

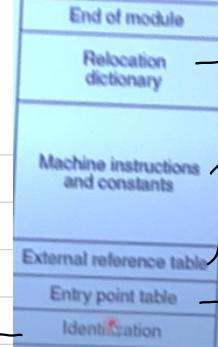
* Traducción VS Interpretación

- Traductor → programa q convierte un programa de un lenguaje fuente) en otro lenguaje (destino) → Compiladores y ensambladores
- Interpret → programa q ejecuta directa 1 sentencia a varios sentencias
 - 1 sentencia a 1 instrucción de máquina
- un programa de un lenguaje escrito en un lenguaje fuente.

- * **Ensambladores** → programa q traduce un programa escrito en lenguaje
 - ↳ 2 tipos
 - ↳ ensamblador q produce código objeto como salida
 - ↳ 2 pasadas en traducción 1 a 1 a lenguaje de máquina
 - ↳ el código
 - ↳ * En la 1^a pasada del proceso de ensamblado genera una **tabla de símbolos**
 - * Para poder armarlo requiere identificar los símbolos y su ubicación relativa
 - Por ende usa el LC (location counter) → puntero interno q lo inicializa en 0 cuando aparece el 1^{er} byte del código objeto ensamblado. y de su posición relativa
 - * El LC se actualiza cada vez q lee una sentencia y se fija cuál es la **longitud de la instrucción** de máquina q reina directivos. A su vez x cada etiqueta encontrada se fija si está en la tabla de símbolos. Si no está se agrega
 - * En la segunda pasada es la traducción de **todas** las inst de ensamblar en código de máquina y la reserva de esos espacios de memoria de acuerdo a los directivos q le fue indicando al ensamblador
 - * Traduce el mnemónico en el opcode binario y lo usa para determinar el formato de instrucción, su posición y tamaño de el u de los campos de la inst.
 - * Traduce el nombre de operando en el registro o código de memoria apropiado
 - * Traduce el valor inmediato en un string binario en la inst
 - * Traduce los referencias a etiquetas en el valor apropiado de LC usando la tabla de símbolos
 - * Setear otros bits necesarios en la codificación de la inst
- * **Código objeto** → representación en lenguaje de máquina del código fuente
 - Formatos extendidos
 - ↳ OMF (object module) creado x un compilador o ensamblador y es luego transformado en código ejecutable x el linkeditor
 - ↳ COFF (common object file format)
 - ↳ ELF (executable and linkable format)

estructura interna

nombre / longitudes / header / pattern de bytes con el
contenido del módulo q se inicializa al archivo



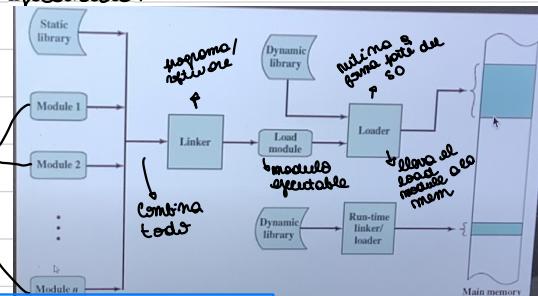
lenta de direcciones a ser
reubicados

o.0 / código ensamblado y constantes
tabla de referencias externas → lista de simbolos usados en el módulo pero definidos fuera de él y sus referencias en el código

tabla de punto de entrada → lista de simbolos q pueden ser referenciados desde otros módulos

* **Linker** → programa utilitario q combina 1 o + archivos con código objeto en un único archivo q contiene código ejecutable o cargable

* **Loader** → rutina de programa q copia un ejecutable a la memoria principal para ser ejecutado.



antes se realizó el código
en forma de objetos
ensamblado

* ¿Porque tiene q existir un linker?

o direcciones externas (quiero usar algo q yo no programé): algunas etiquetas q yo menciono en mi código ensamblado pero q no esté definido su funcionamiento dentro de mi código

o reubicabilidad → q un programa ejecutable puede ubicarse en cualquier área de la memoria principal
↳ no se sabe q otro programa habrá en memoria a lo mejor

* **Linking**

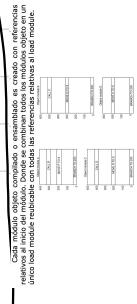
→ **estático**: * el módulo objeto compilado o ensamblado es leído con referencias relativas al inicio del módulo.

* Se combinan todos los módulos objetos en un único load module

reubicable con todas las referencias relativas al load module

* generación del load module:

- ① construye tabla de todos los módulos objeto y sus longitudes
- ② asigna dirección base a cada módulo a la dirección q calculó otros de esa tabla
- ③ suma todos los instrucciones q refieren a memoria y les suma una constante de reubicación igual a la dirección de inicio de un módulo objeto.



④ Busca todos los init q referencia a otros procedimientos e inserta su direc.

↳ **dynamic**: * Se difiere la linkeditión de algún modulo hasta luego de la creación del load module.

↳ **load time dynamic linking**:

① Se levanta a memoria el load module

② Cualquier referencia a un modulo externo hace q el loader busque ese modulo lo cargue y cambie la referencia a una direc relativa desde el inicio del load module

*Ventajas: • Facilita la actualización de versión del modulo externo porque no hay q recompilar.

• El sistema operativo puede cargar y compartir una única versión del modulo externo.

• Facilita la creación de modulos de linkeditón dinámico a los programadores

↳ **Run time dynamic linking**:

* Se pospone el linkeditón hasta el tiempo de ejecución

* Se mantienen las referencias a modulos externos en el programa cargado

* Cuando efectivamente se invoca al modulo externo, el sistema operativo lo lanza, lo carga y linkea al modulo llamador

* Ventajas:

• No ocupa memoria hasta q lo necesito

* **loading**

↳ **loading dinámico en tiempo de ejecución**: a la acción q el loader ejecuta cuando carga un modulo en la memoria principal en el cual el loader no hace nada con los direcs de memoria los dejó tal cuas q los dejó el linkeditón (todos relativos a un LC=0) y a medida q los init se ejecutan null se resuelven los direcs absolutos q el compilador generó para q el hardware

todos los direcs relativos
están relativos a un entorno de ese segmento

↳ **loading x registro base**: Arquitecturas q usan registro base para el direcciónamiento

Se origina un valor para el registro base asociado a la ubicación en la q se carga el programa en memoria

↳ **Loading reubicable**: El compilador/linker genera direcciones relativos al LC=0. El loader debe sumar un valor x a cada referencia a memoria cuando carga el modulo en memoria. El load module tiene q tener info para saber cuales son las referencias a memoria q modificar (dir de reubicación)

↳ **Loading absoluto**: El compilador genera direcs absolutos. solo se puede cargar en un único espacio de memoria.

INTEL

○ ISA (Instruction Set Architecture)

REGISTROS:

- A algunos se puede operar de manera directa:
- Generales:
- Hay 4 tipos:
 - **Acumulador**: operando de instrucciones aritméticas y lógicas : RAX/64 - EAX/32 - AX/16 - AL/8
 - **Base**: direccionamiento de operandos: RBX/64 - EBX/32 - BX/16 - BL/8
 - **Contador**: operaciones aritméticas o de string: RCX/64 - ECX/32 - CX/16 - CL/8
 - **Data**: operaciones que requieren duplas de registros: RDX/64- EDX/32 - DX/16 - DL/8
 - **Indices**:
- Son 2 y tiene usos particulares:
- **Source** : operaciones de manejo de cadenas para apuntar al operando "origen". Manejo de strings / cadenas puntero origen. RSI/64 - ESI/32 - SI/16 - SIL/8
- **Destination**: operaciones de manejo de cadenas para apuntar al operando "destino". Manejo de strings/cadenas puntero destino. RDI/64 - EDI/32- DI/16 - DIL/8
- Pila:
- Almacenar datos de una forma . Ultimo elemento que ingresa es el primero en salir LIFO.
- Base pointer: dirección de memoria de la base de la pila . RBP/64 - EBP/32 - BP/16 - BPL/8
- Stack pointer: dirección de memoria del tope de la pila . RSP/64 - ESP/32 - SP/16 - SPL/8
- Mientras que otros registros operan de manera indirecta:
- Instrucción:
- Instruction Pointer: Actúa como registro contador de programa (PC) y contiene la dirección efectiva de la instrucción siguiente que se ha de ejecutar. RIP/64- EIP/32- IP/16
- Control
- Flags: se usa para almacenar el estado general de la CPU; indica el resultado de la ejecución de cada instrucción. Rflags/64 - Eflags/32 - flags/16

Modos de Direccionamiento :

- **Implícito**: El dato está implícito en el código de operación. → C8H
- **Registro**: El dato está en un registro. → mov RAX, 5
- **Inmediato**: El dato está dentro de la instrucción → mov RAX, 5
- **Directo**: El dato está en memoria referenciado por el nombre de un campo
- **Registro Indirecto**: El dato está en memoria apuntado por un registro base o índice. → mov EAX, [EBX]
 mov EAX, [ESI]
- **Registro Relativo**: El dato está en memoria apuntado por un registro base o índice más un desplazamiento. → mov RAX, [RBX + 4]
 mov RAX, [vector + RSI]
 mov RAX, [vector + RDI]
- **Base + Índice**: El dato está en memoria apuntado por un registro base más un registro índice.
- **Base Relativo + Índice**: El dato está en memoria apuntado por un registro base más un registro índice más un desplazamiento. →
 mov RAX, [RBX + RDI + 4]
 mov RAX, [vector + RBX + RDI]

Tipos de dato

- **Numérico Entero:** Binario de punto fijo con Signo.
- **Numérico Decimal:** Binario de punto Flotante IEEE.
- **Caracteres:** ASCII

Memoria

- **Celda de Memoria:** 1 Byte
- **Palabra :** 2 Bytes
- **Doble Palabra :** 4 Bytes
- **Cuádruple Palabra :** 8 Bytes

Endiannes

- Es el método aplicado para almacenar datos mayores a un byte en una computadora respecto a la dirección que se le asigna a cada uno de ellos en la memoria.
- Existen 2 métodos:
 - **Big-Endian:** determina que el orden en la memoria coincide con el orden lógico del dato. “el dato final en la mayor dirección” Ej. IBM Mainframe
 - **Little-Endian :** es a la inversa, el dato inicial para la lógica se coloca en la mayor dirección y el dato final en la menor. “el dato final en la menor dirección” Ej. Intel

○ Ensamblador

Estructura de un programa :

```
section .data ;variables con contenido  
section .bss ;variables sin contenido  
section .text  
main:  
    ;codigo fuente  
ret
```

Definición y reserva de campos de memoria : (puesto a practica con todos los ej de intel)

○ Instrucciones

Transferencia y Copia

- **MOV op1,op2**
Copia el valor del 2do operando en el primer operando

- **CMP** op1,op2

Compara el contenido del op1 contra el op2 mediante la resta entre los dos operando sin modificarlos

- **JMP** op

Bifurca a la dirección indicada del operando.

- **Jcc** op

Bifurca a la dirección indicada del operando si se cumple la condición.

- **ADD** op1,op2

Suma los valores de los dos operando (binarios de punto fijo con signo) dejando el resultado en el primero.

- **SUB** op1,op2

Resta los valores de los dos operando (binarios de punto fijo con signo) dejando el resultado en el primero.

- **INC** op

Suma uno al operando (binarios de punto fijo con signo)

- **DEC** op

Resta uno al operando (binarios de punto fijo con signo)

- **IMUL** op

Multiplica el contenido de los operandos y almacena el resultado en el primero. Ambos operandos deben tener la misma longitud . Si el resultado no entra en el operando 1, se trunca.

Los operandos son interpretados como binario de punto fijo CON signo

- **MUL** op

Igual que IMUL pero los operandos son interpretados como binario de punto fijo SIN signo

- **IDIV** op

RDX:RAX/op resto en RDX y cociente en RAX . Los operandos son interpretados como binario de punto fijo CON signo

- **DIV** op

Igual que IMUL pero los operandos son interpretados como binario de punto fijo SIN signo

- **G CWD**

Convierte la word almacenada en AX a una double-word en DX:AX

- **CWDE**

Convierte la word almacenada en AX a una double-word en EAX

- **CDQE**

Convierte la doble-word almacenada en EAX a una quad-word en RAX

- **CBW**

Convierte el byte almacenado en AL a una word en AX.

- **NEG** op

Realiza el complemento a 2 del operando, es decir, le cambia el signo

- **LOOP** op

Resta 1 al contenido del registro RCX y si el resultado es 0, bifurca al punto indicado por el operando, sino continua la ejecución en la instrucción siguiente.

El desplazamiento al punto indicado debe estar en un rango entre -128 a 127 bytes (*near jump*)

- **CALL** op

Almacena en la pila la dirección de la instrucción siguiente a la call y bifurca al punto indicado por el operando.

- **RET**

Toma el elemento del tope de la pila que debe ser una dirección de memoria (generalmente cargada por una call) y bifurca hacia la misma

- **AND op1,op2**

Ejecuta la operación lógica AND entre el operando 1 y 2 dejando el resultado en el 1

- **OR op1,op2**

Ejecuta la operación lógica OR entre el operando 1 y 2 dejando el resultado en el 1

- **XOR op1,op2**

Ejecuta la operación lógica EXCLUSIVE OR entre el operando 1 y 2 dejando el resultado en el 1

- **NOT op**

Ejecuta la operación lógica NOT en el operando

- **LEA op1,op2**

Copia en el operando 1 (un registro) la dirección de memoria del operando 2.

- **MOVSB**

Copia el contenido de memoria apuntado por RSI (origen/source) al apuntado por RDI (destino/destination). Copia tantos bytes como los indicados en el registro RCX

- **CMPSB**

Compara el contenido de memoria apuntado por RSI (origen/source) con el apuntado por RDI (destino/destination). Compara tantos bytes como los indicados en el registro RCX

- **PUSH op**

Inserta el operando (de 64 bits) en la pila. Decrementa (resta 1) el contenido del registro RSP

- **POP op**

Elimina el último elemento insertado en la pila (de 64 bits) y lo copia en el operando. Incrementa (suma 1) al contenido del registro RSP

Tablas

- Tira de bytes en memoria destinada a usar como una estructura de Vector o Matriz
- Posicionamiento en el elemento i de un vector : $(i - 1) * \text{longitudElemento}$
- Posicionamiento en el elemento i,j de una matriz : $(i-1)*\text{longitudFila} + (j-1)*\text{longitudElemento}$.
 $\text{longitudFila} = \text{longitudElemento} * \text{cantidadColumnas}$

Validación

- Código destinado a verificar que los datos de un programa provenientes del exterior (teclado, archivos) cumplen las condiciones esperadas y/o necesarias.
- Clasificación según el contenido del dato:
 - **Lógica:** que se adecúe al significado lógico Ej: Dia de la semana: lunes, martes, miércoles, etc
Respuesta "S" o "N"
 - **Física:** que el dato esté en un formato particular Ej: Campo en formato empaquetado .Fecha en formato DD/MM/AA Clasificación según el mecanismo aplicado
 - **Por valor:** comparar contra uno o varios valores válidos
 - **Por Rango:** comparar que esté dentro de un rango continuo válido
 - **Por tabla:** buscar que exista en una tabla de valores válidos

FUNCIONES DE C:

- **puts**

Imprime un string hasta que encuentra un 0 (cero binario). Agrega el carácter de fin de línea a la salida

- **printf**

Convierte a string cada uno de los parámetros y los imprime con el formato indicado por pantalla

- **gets**

Lee una serie de caracteres ingresados por teclado hasta que se presiona 'enter' y los almacena en el campo en memoria indicado por parámetro. Agrega un 0 binario al final.

- **sscanf**

Lee una serie de datos desde un string y, de ser posible, los guarda en el formato indicado para cada uno. Retorna la cantidad de datos que se convirtieron correctamente

- **fopen**

Abre el archivo especificado en *fileName*, en el modo especificado en *mode*. Retorna un id de archivo o un código de error (valor negativo).

- **fgets**

Lee los siguientes *size* bytes (o hasta encontrar el fin de línea) del archivo identificado por *fp* y los copia en *s*. Retorna la dirección de *s* o un código de error.

- **fread**

Lee los siguientes *n* bloques de tamaño *size* bytes del archivo identificado por *fp* y los copia en *p*. Retorna la cantidad de bloques leídos o un código de error.

- **fputs**

Copia los bytes apuntados por *s* hasta encontrar el 0 binario (este último no se copia) en el archivo identificado por *fp*. Retorna un valor negativo en caso de error o fin de archivo.

- **write**

Copia *n* bloques de *size* bytes en el archivo identificado por *fp*. Retorna la cantidad de bloques escritos o un código de error.

- **fclose**

Cierra el archivo identificado por *fp*.

LABORATORIO ARM

Ventajas

- La simplicidad de los procesadores los hace ideales para aplicaciones de baja potencia.
- Dominantes dentro del mercado de la electrónica móvil
 - Microprocesadores pequeños, de bajo consumo y bajo costo ideales para teléfonos móviles, tablets o netbooks.

Características Principales:

- Arquitectura Load/Store : Instrucciones no se ejecutan en memoria, instrucciones entre registros, todas las operaciones son entre registros
- Instrucciones de longitud fija (32 bits)
- Formatos de instrucciones de 3 direcciones :
 - 2 Registros de operandos
 - 1 Registro de resultado
- Ejecución condicional de TODAS las instrucciones.
- Instrucciones de Load-Store de Registros Múltiples.

Organización de Registros

- ARM tiene 16 Registros visibles al programador y un Registro de estado del programa actual, CPSR (Current Program Status Register)
- R0 a R12 son registros de propósito general (32 bits)
 - Usados por el programador para (casi) cualquier propósito sin restricción
- R13 es el Stack Pointer (SP) : manejo de la pila
- R14 es el Link Register (LR) : sirve para anidar llamados dentro de los programas
- R15 es el Program Counter (PC) : apunta a la próxima instrucción a ser ejecutada. Todas las instrucciones tienen longitud de 32 bits y deben estar alineadas a word
- El Current Program Status Register (CPSR) contiene indicadores condicionales y otros bits de estado
- Flags:
 - N = Resultado Negativo del ALU
 - Z = Resultado (Z)cero del ALU
 - C = Operación ALU con aCarreo hacia afuera
 - V = Operación ALU con oVerflow

Organización de Memoria

- Máximo: ~~2³²~~ bytes de memoria
- Word = 32-bits@ Half-word = 16 bits
- Words están alineadas en posiciones divisibles por 4
- Half-words están alineadas en posiciones pares

Estructura de un Programa

- La forma general de una línea en un módulo ARM es: label <espacio> opcode <espacio> operandos <espacio> @ comentario
- Cada campo debe estar separado por uno o más espacios.
- Las instrucciones no empiezan en la primer columna, dado que deben estar precedidas por un espacio en blanco, incluso aunque no haya label.
- ARM aceptará líneas en blanco para mejorar la claridad del código

Interrupciones de Software

- Una forma de leer información y mostrar resultados
 - Instrucción swi que provoca una interrupción de software
 - Para saber que acción debe realizar el SO :
 - Recibe un entero que dice qué hacer
 - El SO también puede leer los registros para obtener información adicional si es necesario
- SWI Codes :
 - Imprimir un entero :
 - Operando 0x6B
 - Registro r1 contiene el entero a imprimir
 - El registro r0 contiene dónde imprimirlo
 - Leer entero desde un Archivo
 - Operando 0x6c
 - R0: manejador de archivo
 - R0: entero
 - Obtener el tiempo actual (ticks)
 - Operando 0x6d
 - R0: número de ticks (milisegundos)
 - Leer string desde un Archivo
 - Operando 0x6a
 - R0: manejador de archivo R1: dirección destino R2: max bytes a almacenar
 - R0: número de bytes almacenados
 - Escribir string
 - Operando 0x69
 - R0: manejador de archivo o Stdout (1) R1: dirección de un string terminado en null
 - Cerrar Archivo
 - Operando 0x68
 - R0: manejador de archivo
 - Abrir Archivo (Modo 0: Input / Modo 1: Output / Modo 2: Append)
 - Operando 0x66
 - R0: dirección de un string terminado en null con el nombre del archivo R1: Modo
 - R0: manejador de archivo (-1 si el archivo no abre)
 - Desasignar todo Bloque de Memoria
 - Operando 0x13
 - Asignar Bloque de Memoria
 - Operando 0x12
 - R0: tamaño del Bloque en bytes
 - R0: dirección del Bloque
 - Mostrar string por consola
 - Operando 0x02
 - R0: dirección de un string terminado en null
 - Mostrar carácter por consola
 - Operando 0x00
 - R0: el carácter
 - fin del programa/Detener la ejecución :
 - Operando 0x11

Secciones del programa

- `.text` especifica la sección de código
- `.data` especifica la sección de variables

Stack + Subrutinas

- Cuando hacemos llamados anidados Para que no se pierda la linea de ejecución debemos almacenar el estado del procesador. Esto podemos hacerlo mediante el uso de la pila :
- **Load Multiple** : `LDM{addr_mode}{cond} Rn{!}, reglist{^}` - Método para recuperar los valores que anteriormente le habíamos hecho push a la pila
- **Store Multiple** : `STM{addr_mode}{cond} Rn{!}, reglist{^}` - Escribe datos en una pila / push
- instrucciones de transferencia de bloques.
- 4 maneras de acceder a la pila:
 - **FD = Full Descending** --> Esta es la mas usada : a medida que uno va a pilando va insertando los valores en memoria descendientemente en cuanto a las direcciones de memoria.
 - `STMFD/LDMFD = STMDB/LDMIA` (nomenclaturas)
 - **ED = Empty Descending**
 - `STMED/LDMED = STMDA/LDMIB`
 - **FA = Full Ascending**
 - `STMFA/LDMFA = STMIB/LDMA`
 - **EA = Empty Ascending**
 - `STMEA/LDMEA = STMIA/LDMDB`

Transferencia de datos de un registro

`ldr @ Load Word`

`str @ Store Word`

`ldrb @ Load Byte`

`strb @ Store Byte`

`ldrh @ Load Halfword`

`strh @ Store Halfword`

`ldrsh @ Load Signed Byte (load and extent sign to 32 bits)` `ldrsh @ Load Signed Halfword (load and extent sign to 32 bits)`

Instrucciones Aritméticas

Add

`ADD{cond}{S} Rd, Rn, <Oprnd2>`

Subtract --> Agarra el segundo operando y le resta el tercero y lo guarda en el primero

`SUB{cond}{S} Rd, Rn, <Oprnd2>`

Reverse subtract --> Agarra el tercer operando y le resta el segundo y lo guarda en el primero (al revés de la resta común)

`RSB{cond}{S} Rd, Rn, <Oprnd2>`

Multiply

`MUL{cond}{S} Rd, Rm, Rs`

Instrucciones Lógicas

And

AND{cond}{S} Rd, Rn, <Oprnd2>

Exclusive Or

EOR{cond}{S} Rd, Rn, <Oprnd2>

Or

ORR{cond}{S} Rd, Rn, <Oprnd2>

Shifts

- Si es de lógica se desplaza hacia la Der en 1 bit y se completa por la izq con 0
- Si es de aritmética se desplaza a la der en 1 bit y se completa por la izq con 0 si el signo es + o con 1 si el signo es negativo -

- En este caso se modifica 1 de los operandos
- Logical Shift Left (multiplica por potencias de 2)

MOV R0, R1, LSL #1

- Logical Shift Right (divide por potencia de 2).

MOV R0, R1, LSR #1

- Arithmetic Shift Right (divide por potencia de 2) preservando el bit de signo

MOV R0, R1, ASR #1

Barrel Shifter

- Cuando se especifica que el segundo operando es un registro shifteado, la operación del Barrel Shifter es controlada por el campo Shift en la instrucción.
- Este campo indica el tipo de shift a realizar.
- La cantidad de bits a shiftear puede estar contenida en un campo inmediato o en el byte inferior de otro registro (que no sea el R15)

Step into: salta a la instr q debe saltar cuando se ejecuta

Step over: salta a la Inst siguiente en linea

Instrucción Compare (cmp)

- Resta dos operandos y descarta el resultado. Sin embargo, se setean los bits de estado (por ej.: acarreo, cero, etc.).
- Si Los operandos son iguales --> (resultado es cero)
- Si 1º operando < 2º operando --> (resultado es negativo)

Ejecución condicional

- La instrucción es ejecutada sólo si el estado actual del flag del código de condición del procesador satisface la condición especificada en los bits b 31- b 28 de la instrucción
- ARM permite que las instrucciones se ejecuten condicionalmente a los valores de los bits de estado . Si coinciden se ejecuta si no coinciden no se ejecuta
 - movmi r0, #42 --> mover si el bit negativo está encendido
 - movpl r0, #42 --> mover si el bit negativo **no** está encendido
 - moveq r0, #42 --> mover si el bit cero está encendido
 - movne r0, #42 --> mover si el bit cero **no** está encendido
- Esta característica elimina la necesidad de utilizar muchas bifurcaciones.
- El costo en tiempo de no ejecutar una instrucción condicional es frecuentemente menor que el uso de una bifurcación o llamado a una subrutina que, de otra manera, sería necesaria.
- Esto mejora la densidad del código y la performance reduciendo el número de instrucciones de bifurcación.

Seteo de Códigos de Condición

- Las operaciones de procesamiento de datos no afectan a los *condition flags*. Para que los *condition flags* se vean afectados, el bit S de la instrucción necesita estar seteado. Esto se hace agregando el sufijo S a la instrucción (y a cualquier código de condición).

Instrucción Branch (b)

- Branch (b) se usa para saltar a código marcado con una etiqueta. El código puede ser etiquetado, al igual que los datos.

Branch condicional

- Punto clave: la instrucción b puede ser ejecutada condicionalmente

Cuando usamos ejecución condicional con branches y cuando ejecución condicional con instrucciones?

- Para traducir sentencias condicionales complejas es más conveniente usar branches etiquetados
- Requerido para condiciones anidadas
- Las instrucciones ejecutadas condicionalmente son más útiles condiciones simples
- En general: hasta tres instrucciones esta bien usar instrucciones condicionales

Loops

- Se utiliza la instrucción Branch con saltos al inicio o al final de un loop
- Para hacer un loop hay q tener una bifurcación condicional y una bifurcación que vuelve para atrás en el código

Enteros en memoria

- La directiva .word define un entero de 32 bits en memoria, así como .asciz define una cadena en memoria

Leer enteros desde memoria

- Paso 1: usar ldr para cargar la dirección en un registro
- Paso 2: usar ldr con [] para cargar el valor en la memoria

Almacenar enteros en memoria

- Paso 1: usar ldr para cargar la dirección en un registro
- Paso 2: usar str para almacenar un valor en esa dirección

Definiendo Arrays

- Única diferencia con las variables: Se especifican múltiples valores con la directiva .word

Accediendo Arrays

- Enfoque básico: incrementar la dirección de memoria
 - Modo de direccionamiento=Cómo el procesador accede a algo
 - El offset puede aplicarse
- Antes de que se realice la transferencia: Direccionamiento Pre-indexado
 - Puede auto-incrementarse el Registro Base agregando ! al final de la instrucción.
 - Despues de que se realice la transferencia: Direccionamiento Post-indexado
 - Causando que el Registro Base se vea auto-incrementado

Direccionamiento Pre-indexado

Modo pre-indexado

- La dirección efectiva del operando es la suma offset de los contenidos del registro base Rn y un offset.

Modo pre-indexado con reescritura

- La dirección efectiva del operando se genera de la misma manera que en el modo Pre-indexado pero la dirección efectiva se escribe también en Rn.

Modo post-indexado

- La dirección efectiva del operando es el contenido de Rn. El desplazamiento se agrega a esta dirección y el resultado se escribe de nuevo en Rn.

Registro indirecto

- Acceso a la memoria realizado a través de una dirección en un registro
- Offset de 4 bytes porque una palabra es de 4 bytes
- Ej:

```
.word 32, 65, 76
.text
ldr r0, =arr
ldr r1, [r0]
add r0, r0, #4
ldr r2, [r0]
add r0, r0, #4
ldr r3, [r0]
```

Registro indirecto con post-incremento

Códigos de Condición

Code [31:28]	Mnemonic	Interpretación	Status flag state required
0000	EQ	Igual / Igual a cero	Z seteado
0001	NE	Distinto	Z vacío
0010	CS/HS	Carry seteado / ≥ sin signo	C seteado
0011	CCLO	Carry vacío / < sin signo	C vacío
0100	MI	Menor / negativo	N seteado
0101	PL	Mayor / positivo o cero	N vacío
0110	VS	Overflow	V seteado
0111	VC	No overflow	V vacío
1000	HI	> sin signo	C seteado y Z vacío
1001	LS	≤ sin signo	C vacío y Z seteado
1010	GE	≥ con signo	N igual a V
1011	LT	< con signo	N distinto de V
1100	GT	> con signo	Z vacío y N igual a V
1101	LE	≤ con signo	Z seteado o N distinto de V
1110	AL	Siempre	Cualquiera
1111	NV	Nunca	Ninguno

Bifurcaciones condicionales

Mnemonic	Interpretación	Aplicación
B	Incondicional	Siempre bifurcar
BAL	Siempre	Siempre bifurcar
BEQ	Igual	Comparación igual o resultado es cero
BNE	Distinto	Comparación distinta o resultado no es cero
BPL	Positivo	Resultado positivo o cero
BMI	Negativo	Resultado negativo
BCC	Flag Carry vacío	Operación aritmética no resultó en acarreo hacia afuera
BLO	Menor	Comparación sin signo con resultado menor
BCS	Flag Carry seteado	Operación aritmética resultó en acarreo hacia afuera
BHS	Mayor o igual	Comparación sin signo con resultado mayor o igual
BVC	Flag Overflow vacío	Operación de entero con signo: resultado sin overflow
BVS	Flag Overflow seteado	Operación de entero con signo: resultado con overflow
BGT	Mayor	Comparación de entero con signo: mayor

BGE	Mayor o igual	Comparación de entero con signo: mayor o igual
BLT	Menor	Comparación de entero con signo: menor
BLE	Menor o igual	Comparación de entero con signo: mayor o igual
BHI	Mayor	Comparación de entero sin signo: mayor
BLS	Menor o igual	Comparación de entero sin signo: menor o igual

modos de direccionamiento

Nombre	Nombre Alternativo	Ejemplos
Registro a registro	Registro directo	mov r0, r1
Absoluto	Directo	ldr r0, mem
Literal	Inmediato	mov r0, #15 add r1, r2, #12
Indexado, base	Registro indirecto	ldr r0, [r1]
Pre-Indexado, base con desplazamiento	Registro indirecto con offset	ldr r0, [r1, #4]
Pre-indexado, autoindexado	Registro indirecto con pre-incremento	ldr r0, [r1, #4]!
Post-indexado, autoindexado	Registro indirecto con post-incremento	ldr r0, [r1], #4
Doble registro indirecto	Registro indirecto indexado	ldr r0, [r1, r2]
Doble registro indirecto escalado	Registro indirecto indexado escalado	ldr r0, [r1, r2, lsl #2]
Relativo al PC		ldr r0, [PC, #offset]

U5 → Componentes de un computador

* Segun la arquitectura de Von Neuman → el Computador Conceptual // se podía describir con 3 unidades funcionales → CPU / Unidad de procesamiento central
la entrada
y salida
memoria

MEMORIA:

Se la suele asociar de manera inmediata con RAM (random access memory)

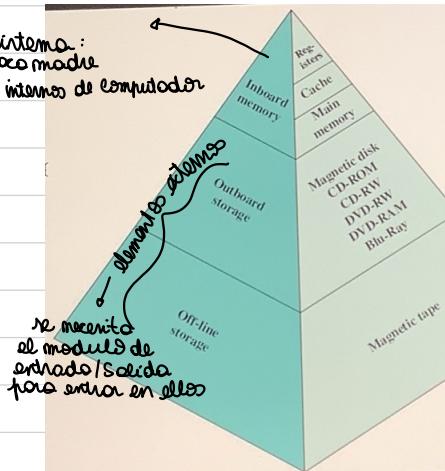
- Varios elementos que componen el sistema
- Tiene cualidades diversas que tiene que ver con tecnologías organización performance y costos .
- Jerarquía de subsistemas de memoria:
 - Internos al sistema (accedidos directamente por el procesador)
 - Externos al sistema (accedidos por el procesador a través de un módulo de E/S)
- Tres características a tener en cuenta
- ○ Capacidad

○ Tiempo de acceso

○ Costo

- No hay un elemento de los 3 que sea mas importante , relación de ~~programación entre estas~~ características

memoria interna del sistema :
física dentro de la placa madre
donde están los circuitos internos de computador



accesibles al programador y cualquier otro registro que la computadora utilice para almacenar información . Tiene la capacidad del tamaño de la arquitectura

- Memoria cache: Memoria que permite almacenar temporalmente información . Sirve para acelerar la performance del computador. Se guarda temporalmente copias de la memoria principal y si las búsquedas que se hacen de esa info están almacenadas en esa cache se van a resolver más rápido q si fuéramos a la memoria principal. Capacidad de almacenamiento muchísimo mayor que un registro.

- Memoria principal = ram : En la cual se almacenan las instrucciones y los datos a la hora de ejecutarlos en cualquier computadora de arquitectura von neuman . Mas lenta el acceso que a la cache , mas barata y mayor capacidad de almacenamiento .

Jerarquía de memoria A medida que se baja de la pirámide:

- Costo por bit decreciente
- Capacidad creciente
- Tiempo de acceso creciente (es decir cada vez mas lento)
- Frecuencia de acceso de la memoria por parte de procesador decreciente

Sistema de memoria

- Características
- Locación
 - Interna
 - Registros
 - Memoria interna para unidad de control
 - Memoria Cache
 - Externa
- Dispositivos de almacenamiento periféricos (discos, cintas, etc.)
 - Capacidad
 - Bytes / Palabras (memoria interna)
 - Bytes (memoria externa)
 - Unidad de transferencia = q info transfiero
 - Número de líneas eléctricas del módulo de memoria, típicamente el tamaño de palabra o 64, 128 o 256 bits (memoria interna)
 - Bloques (memoria externa)
 - Métodos de acceso de unidades de datos
 - Acceso secuencial : la Info se va guardando de a bloques uno detrás del otro y la cinta va girando en el carrete por ende si quiero acceder a algo mas adelante que el carrete tengo que saltarme la info que esta antes, no puedo saltarlo por ende es secuencia
 - Unidades de datos: registros (records)
 - Acceso lineal en secuencia
 - Se deben pasar y descartar todos los registros intermedios antes de acceder al registro deseado
 - Tiempo de acceso variable
 - Ej. cintas magnéticas
 - Acceso directo
 - Dirección única para bloques o registros basada en su posición física
 - Tiempo de acceso variable - Ej. discos magnéticos
 - Acceso aleatorio
 - (*) Cada posición direccionable de memoria tiene un mecanismo de direccionamiento cableado físicamente (caminito de acceso a cada byte de esa memoria que es independiente del otro).
 - Tiempo de acceso constante porque (*), independiente de la secuencia de accesos anteriores
 - Ej. memoria principal y algunas memorias cache
 - Acceso asociativo
 - Tipo de acceso aleatorio por comparación de patrón de bits
 - La palabra se busca por una porción de su contenido en vez de por su dirección
 - Cada posición de memoria tiene un mecanismo de direccionamiento propio
 - Tiempo de acceso constante, independiente de la secuencia de accesos anteriores o su ubicación
 - Ej. memorias cache
 - Parámetros de performance
 - Tiempo de acceso (latencia)
 - Memoria de acceso aleatorio: tiempo necesario para hacer una operación de

lectura o escritura

- Memorias sin acceso aleatorio: tiempo necesario para posicionar el mecanismo de lectura/escritura en la posición deseada

- Tiempo de ciclo de memoria

- Memorias de acceso aleatorio: tiempo de acceso más el tiempo adicional necesario para que una nueva operación pueda comenzar

- Tasa de transferencia

- Tasa con la cual los datos son transferidos dentro o fuera de la unidad de memoria

- Memorias de acceso aleatorio: $1/\text{Tiempo de ciclo de memoria}$

- Memorias sin acceso aleatorio: $T = T + n/R \text{ } nA$ donde

Tn = Tiempo promedio para leer o escribir n bits T = Tiempo promedio de acceso A

n = Número de bits R = Tasa de transferencia, en bits por segundo (bps)

- Tipos físicos

- Memorias semiconductoras (memoria principal y cache) SSD

- Memorias de superficie magnética (discos y cintas)

- Memorias ópticas (medios ópticos)

- Características físicas

- Memorias volátiles: se pierde su contenido ante la falta de energía eléctrica (Ej. algunas memorias semiconductoras). Desenchufo la maquina y se pierde todo

- Memorias no volátiles: no se necesita de energía eléctrica para mantener su contenido (Ej. memorias de superficie magnéticas y algunas memorias semiconductoras). Persiste

- Memorias de solo lectura: (ROM – Read Only Memory) no se puede borrar su contenido (Ej. algunas memorias semiconductoras)

- Dynamic RAM (DRAM)

- Celdas que almacenan datos como carga en capacitores (bit 0 o 1 si hay presencia o ausencia de carga en el capacitor)
- Requieren refrescar su carga periódicamente para mantener los datos almacenados
- Se suelen usar en memoria principal

- Static RAM (SRAM)

- Los valores binarios se almacenan usando compuertas lógicas flip-flop

- Son más complejas y grandes que las DRAM

- Son más rápidas que las DRAM

- Se suelen usar en memoria cache

MEMORIA CACHE

- Principio de localidad de referencia —> por esto tiene sentido q la cache sea efectiva

- "Durante la ejecución de un programa, las referencias a memoria que hace el procesador tanto para instrucciones como datos tienden a estar agrupadas (zonas contiguas de memoria)" (Ej. loops, subrutinas, tablas, vectores) (agrupación de instrucciones)

- Memoria semiconductora más rápida (y costosa) que la principal

- Se ubica entre el procesador y la memoria principal

- Permite mejorar la performance general de acceso a memoria principal

- Contiene una copia de porciones de memoria principal

- Cómo funciona :

- CPU trata de leer una palabra de la memoria principal

- Se chequea primero si existe en la memoria cache.

- Si es así se la entrega al CPU

- Sino se lee un bloque de memoria principal (número fijo de palabras), se incorpora a la cache y la palabra buscada se entrega al CPU

- Por el principio de localidad de referencia es probable que próximas palabras buscadas estén dentro del bloque de memoria subido a la cache
 - La memoria cache se nutre del principio de localidad porque cuando el CPU pide acceder a un proximo dato de donde esta ejecutando la instrucción de maquina y uno de los operando es un área de memoria . Cuando el CPU ejecuta esa búsqueda de esos datos en los sistemas que hay memoria cache , no accede a la memoria RAM sino a la cache y si este no tiene el dato la cache le pide a la RAM la posicion de memoria y las 10 palabras que le siguen y luego le devuelve al CPU lo q pidió. Si el CPU en la próxima le pide alguna posición siguiente a la anterior la memoria cache ya la tiene y entonces el acceso es mucho mas rápido . Mas perforaste mas accesos a cache que a RAM
 - Estructura sistema cache/memoria principal
 - Memoria principal
 - 2^n palabras direccionables (dirección única de n -bits para cada una)
 - Bloques fijos de K palabras cada uno (M bloques)
 - Cache
 - m bloques llamados líneas
 - Cada línea contiene:
 - K palabras
 - Tag (conjunto de bits para indicar qué bloque está almacenado, usualmente una porción de la dirección de memoria principal)
 - Bits de control (Ej. bit para indicar si la línea se modificó desde la última vez que se cargó en la cache)
- Administración de Memoria
 - Sistema Operativo
 - "Software que administra los recursos del computador, provee servicios y controla la ejecución de otros programas"
 - Algunos servicios que provee
 - Schedule de procesos
 - Administración de memoria
 - Monitor
 - Parte residente del Sistema Operativo
 - Uniprogramación
 - Un solo proceso de usuario en ejecución a la vez
 - La memoria de usuario está completamente disponible para ese único proceso
 - Uso del procesador a lo largo del tiempo
 - Run: Tiempo efectivo de uso del CPU
 - Wait: Tiempo ocioso del CPU esperando E/S (*Idle time*)
 - Administración de memoria simple
 - Sistema con uniprogramación
 - Se divide la memoria en dos partes
 - Monitor del S.O.
 - Programa en ejecución en ese momento
 - Ventajas:
 - Simplicidad
 - Desventajas:
 - Desperdicio de memoria
 - Desaprovechamiento de los recursos del computador
 - Ej. MS-DOS, iPhone OS v1-3, IBM OS/PCP (Primary Control Program)

- Multiprogramación
 - Varios procesos de usuario en ejecución a la vez
 - Se divide la memoria de usuario entre los procesos en ejecución
 - Se comparte el tiempo de procesador entre los procesos en ejecución (timeslice)
 - Condiciones de finalización de los procesos:
 - Termina el trabajo
- Se detecta un error y se cancela
- Requiere una operación de E/S (suspensión)
 - Termina el timeslice (suspensión)
 - Administración de memoria por asignación particionada → Primer mecanismo básico para trabajar con multiprocesos a la vez
 - Sistema con multiprogramación
 - La memoria de usuario el SO la divide lógicamente en particiones de tamaño fijo:
 - Iguales → Ningún programa podía excederse del tamaño de la partición, todas las opciones mismo tamaño
- Distintas → + versatilidad + opciones , el SO podía elegir en q partición ubicar el programa en base al tam del programa
 - Ventajas:
- Permite compartir la memoria entre varios procesos
 - Desventajas:
- Desperdicio de memoria
 - Fragmentación interna (dentro de una partición) → si guardo un programa + chico que el tam de la partición desperdicio el resto
 - Fragmentación externa (particiones no usadas) → si el tam de la partición es menor a mi programa no puedo usarla por ende se desperdicia
 - Ej. IBM OS/MFT (Multiprogramming with a Fixed number of Tasks)
 - Administración de memoria por asignación particionada reasignable → Se trabajaba en partición pero se iban alocando de acuerdo a la necesidad de cada proceso
 - Sistema con multiprogramación
 - Swapping → Permite al SO subir o bajar procesos de memoria de acuerdo a la necesidad de gestión q tenga
 - La memoria de usuario se divide en particiones de tamaño variable
 - Compactación para eliminar la fragmentación → SO barría toda la memoria y determinaba cuáles eran los huecos y los compactaba para eliminar la fragmentación
 - Se usa un recurso de hardware (registro de reasignación) para la realocación
 - Realocación dinámica en tiempo de ejecución
 - Ventajas:
 - Permite compartir la memoria entre varios procesos
 - Elimina el desperdicio por fragmentación interna. Con la compactación se elimina / minimiza además la fragmentación externa
 - Desventajas:
 - La tarea de compactación es costosa
 - Ej. IBM OS/MVT (Multiprogramming with a Variable number of Tasks)

- Administración de memoria paginada simple --> Parecido a lo q se usa hoy
- . Sistema con multiprogramación
- . Se divide el address space del proceso en partes iguales (páginas) (ej. IA-32 4KB c/u)
- . Se divide la memoria principal en partes iguales (frames)
- . Hay una tabla de páginas por proceso
- . Hay una lista de frames disponibles
- . Se cargan a memoria las páginas del proceso en los frames disponibles (no es necesario que sean contiguos)
 - . Las direcciones lógicas se ven como número de página y un offset
 - . Se traducen las direcciones lógicas en físicas (*address translation*) con soporte del hardware (MMU – Memory Management Unit)
 - . La paginación es transparente para el programador

- Ventajas:

- . Permite compartir la memoria entre varios

procesos

- . Permite el uso no contiguo de la memoria
- . Minimiza la fragmentación interna (solo existe dentro de la última página de cada proceso)
- . Elimina la fragmentación externa

- Desventajas:

- . Se requiere subir todas las páginas del proceso a memoria
- . Se requieren estructuras de datos adicionales para mantener información de páginas y frames

- Administración de memoria paginada por demanda (memoria virtual) --> En la memoria principal; solo se van a cargar algunas páginas necesarias para la ejecución de un proceso

- . Sistema con multiprogramación
- . Solo se cargan a memoria principal las páginas necesarias para la ejecución de un proceso
- . Cuando se quiere acceder a una posición de memoria de una página no cargada se produce un *page fault*

- El page fault dispara una interrupción por hardware (MMU) atendida por el sistema operativo

- . El sistema operativo (*page fault handler*) levanta la página solicitada desde memoria secundaria (memoria virtual)
- . Si no hay frames libres es necesario bajar páginas a memoria secundaria y reemplazarlas (*page swapping*)

- Algoritmos para reemplazo de páginas (por ejemplo FIFO, First In First Out o LRU, Least Recently Used)

- Thrashing: el CPU pasa más tiempo reemplazando páginas que ejecutando instrucciones

- Ventajas:

- . No es necesario cargar todas las páginas de un proceso a la vez
- . Maximiza el uso de la memoria al permitir cargar más procesos a la vez
- . Un proceso puede ocupar más memoria de la efectivamente instalada en el computador

- Desventajas:

- . Mayor complejidad por la necesidad de implementar el reemplazo de páginas

Ej. Windows 3.x en adelante, Linux

- Administración de memoria por segmentación
- Sistemas con multiprogramación
- Generalmente visible al programador
- La memoria del programa se ve como un conjunto de segmentos (múltiples espacios de direcciones)

- Los segmentos son de tamaño variable y dinámico
- El sistema operativo administra una tabla de segmentos por proceso
- Permite separar datos e instrucciones
- Permite dar privilegios y protección de memoria como por ej. lectura, escritura, ejecución.

(segmentation faults como mecanismos de excepción de hardware para accesos indebidos)

- Las referencias a memoria se forman con un número de segmento y un offset dentro de él. Con ayuda de hardware (MMU – Memory Management Unit) se hacen las traducciones de las direcciones lógicas a físicas

- Se pueden usar para implementar memoria virtual (solo se suben a memoria física algunos segmentos por proceso)

- Ventajas:

- ○ Simplifica el manejo de estructuras de datos con crecimiento
- ○ Permite compartir información entre procesos dentro de un segmento
- ○ Permite aplicar protección/privilegios sobre un segmento fácilmente

- Desventajas:

- ○ Fragmentación externa en la memoria principal por no poder alojar un segmento
- ○ Hardware más complejo que memoria paginada para la traducción de direcciones Ej.

Burroughs Corporation B5000-B6500, IBM AS/400, Intel x86 (por compatibilidad hacia atrás)

- Distintas combinaciones (Ej. Intel Pentium)

- Sin segmentación y sin paginación

- Direcciones lógicas iguales a las físicas. No es útil para multiprogramación. Usado en controladores de alta performance

- Paginación sin segmentación

- La protección y administración de la memoria se hace a través de las páginas.
- Ej. Berkeley UNIX

○ Segmentación sin paginación

- La memoria se ve como una colección de espacios lógicos, con protección a nivel segmentos.

Segmentación con paginación

- Segmentos para controlar el acceso a particiones de memoria.
- Páginas para administrar la locación dentro de los segmentos
- Ej. UNIX System V

Entrada/Salida

- Módulo de E/S
- ¿Qué hace?

- Conecta a los periféricos con la CPU y la memoria a través del bus del sistema o switch central y permite la comunicación entre ellos

- ¿Por qué existe?

- Amplia variedad de periféricos con distintos métodos de operación

- La tasa de transferencia de los periféricos es generalmente mucho más lenta que la de la memoria y procesador

- Los periféricos usan distintos formatos de datos y tamaños de palabra
 - ¿Para qué sirve?
- Oculta detalles de timing, formatos y electromecánica de los dispositivos periféricos porque si el CPU tuviera que hacerlo estaría desperdiciado
 - Interface interna (bus del sistema)
 - Datos
 - Direcciones
 - Control
 - Interface externa (periféricos)
 - Datos
 - Control
 - Estado
 - Funciones
 - Control & Timing
 - Controla flujo de tráfico entre CPU/Memoria y periféricos
 - Comunicación con el procesador - Decodificación de comandos
 - Datos
 - Información de estado
 - Reconocimiento de direcciones
 - Comunicación con el dispositivo
 - Comandos
 - Información de estado
 - Datos
 - Buffering de datos —> Poder resguardar temporalmente los datos hasta que se pueda continuar con el proceso q se esta llevando a cabo
 - Detección de errores
 - Técnicas para operaciones de E/S
 - E/S Programada—> El CPU quiere leer algo del disco rígido de la maquina entonces le va a enviar algún comando de lectura al modulo de entrada y salida y este lo va a procesar . Mientras lo procesa el CPU en forma recurrente va a consultar el estado de esa operación , le va a responder si hay un error si no esta listo o si si le va a enviar al CPU esa información q le pidió leer . El CPU escribe la palabra recibida en memoria . El problema es q el CPU espera q la memoria cache le responda y no hace nada productivo por ende es un desperdicio de performance d ella maquina .
 - E/S manejada por interrupciones —> Cuando el CPU manda la instrucción no se queda esperando q pasa con ese comando sino q automáticamente se reasigna el CPU para que ejecute instrucciones de maquina de cualquier otro proceso q este en esa maquina . Se entera q ocurrió con el proceso anterior mediante el mecanismo de interrupciones (un elemento del software/hardware pueden interrumpir la ejecución de lo q este haciendo el CPU para advertirle q hay algo mas para atender) El CPU se va a enterar del resultado mediante una interrupción . El resto es parecido a la E/S programada.
 - Acceso directo a memoria (DMA) —> EL CPU va a emitir el comando para leer el bloque entero se la va a mandar al DMA (modulo de E/S) mientras va a hacer otra cosa . EL DMA va a gestionar las múltiples lecturas va a resolver la escritura del resultado en la memoria. El DMA genera la interrupción después de resolver .
 - Información enviada por el CPU al DMA
 - Operación (READ o WRITE) vía Línea de Control
 - Dirección del dispositivo vía Línea de Dirección
 - Dirección inicial de memoria para READ o WRITE vía Línea de Datos, almacenado en el address register
 - Cantidad de palabras para READ o WRITE vía Línea de Datos, almacenado en el data count

- Data Count: Cuantas palabras tiene q leer o escribir
- Data register : guarda la info q tiene q escribir/leyo del periférico
- Adress register: va a recibir cual es la dirección inicial del área de memoria q tiene q leer o escribir
- Control Logic: va a recibir los comandos y los va a decodificar o mandar al periférico .
- **Robo de ciclos** → EL CPU va a permitir q el DMA acceda a la mem 3 momentos ; uno antes de q el CPU haga el fetch de la instrucción, otro previo al fetch de un operando y el ultimo justo antes de escribir una instrucción de maquina .

- **Topología:**
- DMA desacoplado unico bus todos conectados al unico bus
- DMA integrado unico bus pero DMA integrado con distintos modelos de entrada y salida .
- DMA E/S tengo un bus de sistema y otro de entrada y salida
- Canales y procesadores de E/S

Canales

- Tienen la habilidad de ejecutar instrucciones de E/S
- La CPU principal no ejecuta instrucciones de E/S
- Las instrucciones de E/S se almacenan en memoria principal
- La CPU le indica al canal de E/S que inicie un programa de canal

- Dispositivo

- Área de memoria para storage

- Prioridad
- Acciones ante errores

Procesadores

- Agregan a los canales memoria propia en vez de usar la memoria principal

Tipos de canales

- Selectores
 - Usa un dispositivo a la vez a través de un controlador de E/S
- Multiplexores

- Trabaja con múltiples dispositivos a la vez (multiplexa los flujos de datos)

INTERRUPCIONES

- ¿Qué son?

“Mecanismos por los cuales otros módulos (E/S, memoria, etc.) **interrumpen el normal procesamiento del CPU”**

- ¿Para qué existen?

“Para mejorar la eficiencia de procesamiento de un computador”

- Clases de interrupciones

Hardware

Soft

- Clases de interrupciones

Hardware (asincrónicas)

E/S

Reloj (timer)

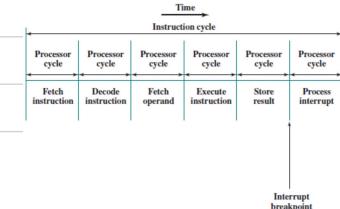
Fallas de hardware

Software

Excepciones de programa

- División por cero
- Acceso indebido a memoria
- Overflow
- Instrucción inválida

Instrucciones privilegia



Ciclo de instrucción > Se ejecutan las interrupciones de principio a fin

- Transferencia de control al S.O → Se ejecutan según su importancia

?

Múltiples interrupciones (ejemplo)

○ Tres dispositivos de E/S

- Línea de comunicación (Prioridad 1)
- Disco (Prioridad 2)
- Impresora (Prioridad 3)

○ Eventos

- T=10 – Interrupción de impresora
- T=15 – Interrupción de línea de comunicación
- T=20 – Interrupción de disco

PROCESADOR

- Arquitectura de procesadores

○ Historia

- Primero orientado al hardware (hasta los '70)
- Luego orientado al software (a partir de los '80)

○ CISC vs RISC

- CISC (Complex Instruction Set Computer)
 - Pocos registros de procesador (especializados), se priorizaba el acceso a memoria
 - Set de Instrucciones amplio
 - Muchas instrucciones para trabajar con memoria
 - Microarquitectura en software/hardware compleja
 - Instrucciones complejas (más de un ciclo de reloj)
 - Varios modos de direccionamiento
 - Muchos tipos de datos
 - Muchos formatos de instrucción (variables o híbridos)
 - Orientado al hardware, compiladores relativamente simples (tamaño de código pequeño)
 - Ejemplos: VAX, Intel x86 (hasta IA-32), Intel-64, IBM Mainframe, Motorola 6
- RISC (Reduced Instruction Set Computer) → Arquitecturas más simples, ejecutar muchas instrucciones más simples es más performante que ejecutar menos pero + complejas
 - Muchos registros de procesador de uso general
 - Set de Instrucciones pequeño
 - Solo acceso a memoria a través de LOAD/STORE
 - Microarquitectura en hardware simple
 - Instrucciones simples (un ciclo de reloj)
 - Pocos modos de direccionamiento
 - Pocos tipos de datos
 - Pocos formatos de instrucción (fijos)
 - Orientado al software, compiladores relativamente complejos (tamaño de código largo)
 - Ejemplos: SPARC, MIPS, ARM, Intel Itanium (IA-64)

- Procesador

- Arquitectura de procesadores

- ¿Cómo se mejora la performance de ejecución de un procesador?

- O aumenta la frecuencia del reloj → genera más ciclos de instrucción por unidad de tiempo o paraleliza la ejecución de instrucciones por ciclo de reloj

○ Ecuación de Performance

$$\text{MIPS rate} = \text{Frecuencia del reloj en MHz} (f) *$$

Instrucciones por ciclo (IPC)

○ Uniprocesadores

○ Taxonomía de Flynn

- SISD (Single Instruction Single Data) (Uniprocesadores)
- SIMD (Single Instruction Multiple Data) vectoriales

- MISD (Multiple Instruction Single Data) (No comercial)

- MIMD (Multiple Instruction Multiple Data)

 ? Limitación de la velocidad del reloj (calor)

○ Paralelismo

 ? Técnicas

 - A nivel instrucción

 - Pipelining

 - Dual pipelining

 - Superscalar

 - Multithreading

 - A nivel procesador

 - Procesadores paralelos de datos - Multiprocesadores

 - Multicomputadores

○ A nivel instrucción

 ? Pipelining (Stages) -> La idea es parallelizar las distintas etapas q tiene la ejecución de una instrucción de maquina

 - Solapa la ejecución de las instrucciones para reducir el tiempo total de una secuencia de instrucciones

 - Ejecuta una instrucción por ciclo de reloj

 - Control de dependencia entre las instrucciones (compilador o hardware) - Ej. Intel 486

 - Riesgo de procesar varias instrucciones a la vez -> q una instrucción requiera de la resolución

de una anterior q aun no haya finalizado

 ? Dual Pipelining -> dos cadenas de ejecución al mismo tiempo

 - Ejecuta dos instrucciones por ciclo de reloj - Ej. Intel Pentium

 - No es una buena idea porque muchas instrucciones al mismo tiempo complejidad mas mas posibilidades de requerir una instrucción q no finalizo o bifurcaciones

 ? Superscalar (múltiples unidades funcionales en la etapa de ejecución)

 - Ejecuta más de una instrucción por ciclo de reloj

 - N-way / N-issue (N entre 3 y 6)

 - Ej. Intel Core

 - Mas velocidad a la etapa posterior a la 3 q es mas lenta por ende rompe con el cuello de botella. La 3 etapa envía múltiples instrucciones q se ejecutan en paralelo

 ? Hardware multithreading

 - Busca incrementar el uso del CPU intercambiando la ejecución entre threads (hilos de ejecución) cuando uno está frenado por alguna causa

 - Thread: Contiene un PC, un conjunto de registros y la pila (stack). Comparten un mismo address space. Se los conoce como "lightweight processes"

 - Proceso: Puede tener uno o más threads, contiene un address space y un estado gestionado por el S.O.

 - El cambio de contexto entre threads es "liviano" (en un mismo ciclo de reloj) en comparación con los procesos, que requieren del S.O.

 - Fine-grained multithreading: se intercambia el uso del procesador entre threads luego de la ejecución de cada instrucción. Ej. Procesadores Intel IA-32

 - Coarse-grained multithreading: se intercambia el uso del procesador entre threads solo luego de algún evento significativo, como puede ser un page fault o un "cache miss". En este último caso se cambia la ejecución a otro thread en vez de esperar el acceso más lento a la memoria principal. Ej. Intel Itanium 2

?

Procesadores paralelos de datos

- Una sola unidad de control
- Múltiples procesadores

?

Métodos

- SIMD – Single Instruction Multiple Data
 - Múltiples procesadores ejecutan la misma secuencia de pasos sobre un conjunto diferente de datos
 - Ej. GPU (Nvidia Fermi GPU)
- Vectoriales
 - Similar a SIMD
 - Registro vectorial: conjunto de registros convencionales que se cargan desde memoria en una sola instrucción.
 - Se opera por pipelining
 - Ej. Intel Core (SSE – Streaming SIMD Extension)

?

Multiprocesadores

- Múltiples CPUs que comparten memoria común - MIMD (Multiple Instruction Multiple data)
- CPUs fuertemente acoplados
- Diferentes implementaciones
 - Single bus y memoria compartida (centralizada) (UMA – Uniform memory access) (SMP – Symmetric multiprocessor)
 - Ej. Intel Core i7
 - CPUs con memoria local y memoria compartida (NUMA – non-uniform memory access)
 - Ej. BBN Butterfly, SGI Origin 2000, Compaq AlphaServer GS320, Intel Itanium 2

?

Multicomputadores

- Computadores interconectados con memoria local (memoria distribuida)
- No hay memoria compartida
- CPUs ligeramente acoplados - Clusters
- MIMD (Multiple Instruction Multiple data)
- Intercambio de mensajes
- Topologías de grillas, árboles o anillos
- Ej. IBM Blue Gene/P

U6 → Almacenamiento Secundario

ALMACENAMIENTO SECUNDARIO

- **Cintas magnéticas**
- Medio
 - **Poliester flexible cubierto de material magnetizable**
 - Carretes abiertos
 - **Paquetes cerrados (cartuchos)**
 - Ancho de cinta entre 0.38 cm (0.25 pulgadas) y 1.27 cm (0.5 pulgadas). Dentro de este el medio estaba preformatoado con distintas pistas (carril dentro del ancho de la cinta en el cual factible q se produjera la grabación a lectura de un bit de información)
 - **Acceso secuencial a la información:** si estoy en el registro 1 y quiero llegar al N tengo que “leer” los N-1 del medio
 - Si quiero leer un registro anterior tengo que rebobinar y volver a buscar el registro
- **Técnicas de grabación**
 - **Grabación en paralelo**
 - Técnica usada originalmente
 - Cabeza de grabación estacionaria
 - Se graban pistas en paralelo a lo largo de la cinta
 - Al principio eran de 9 pistas (8 bits de datos y 1 bit de paridad para detectar errores)
 - Luego fueron 18 (palabra) o 36 (doble palabra) pistas
 - **Grabación en serie**
 - Sistema moderno de grabación
 - Cabeza de grabación estacionaria
 - Se escriben los datos a lo largo de una pista primero hasta llegar al final de la cinta y luego se pasa a otra
 - Grabación en “serpentina”
 - Pueden grabarse n pistas adyacentes en simultáneo (n entre 2 y 8)
 - **Grabación helicoidal**
 - Cabeza de grabación rotatoria
 - Símil video caseteras
 - Evita problema de movimiento veloz de la cinta de las otras técnicas
 - La cinta se mueve en forma lenta mientras que la cabeza rota en forma rápida
 - Las pistas pueden estar más cercanas unas a otras
 - Formatos:
 - DAT/DDS (4mm), AIT (8mm), Exabyte Mammoth (8mm), SAIT (1/2 "), etc.
- **Modos de operación**
 - **Modo start-stop por bloque**
 - Viejo uso de grabación por registro/bloque
 - La cinta se usaba para guardar archivos para procesamiento posterior
 - Se podía actualizar un registro/bloque particular siempre y cuando no cambiara su tamaño
 - Los datos se grababan en bloques físicos
 - Entre los bloques había espacios (IRG – Inter Record Gap) para sincronización de la unidad

○ Modo streaming

- Uso para backup o archivo de información
- No se requiere operación de start-stop por bloque
- No se requiere actualización de bloques particulares dentro de un archivo
- Se escriben archivos completos como un "stream" de datos contiguo
- La información se graba físicamente en bloques pero no se pueden localizar o modificar bloques particulares

- Usos y características

- Fue el primer medio de almacenamiento secundario
- Aun es usado para backup y archivo de información (30 años o más de duración) dado su bajo costo por byte y su capacidad de almacenamiento
- Es el medio más lento de la pirámide de jerarquía de memoria.
- Marcas físicas en las cintas
 - ☐ BOT (Beginning of tape)
 - ☐ FOT (End of tape)

- Discos magnéticos

- Medio

- Plato circular construido de un material no magnético, llamado substrato (aluminio o vidrio), cubierto por un material magnetizable
- Mecanismos de lectura/escritura magnético
 - Cabeza de lectura/escritura única: bobina conductora estática, el disco está girando constantemente debajo de ella
 - Usado en los viejos discos rígidos y floppy disk
 - Escritura: cuando circula electricidad a través de una bobina se produce un campo magnético. Los patrones magnéticos resultantes se graban en la superficie (diferentes patrones para corrientes + y -)
 - Lectura: un campo magnético que se mueve por una bobina produce corriente eléctrica en ella. Cuando la superficie del disco pasa debajo de la cabeza se genera una corriente de la misma polaridad grabada

○ Mecanismos de lectura/escritura magnético

- Cabeza de lectura diferenciada de la de escritura
- Tiene un sensor magneto-resistivo (MR)
- La resistencia eléctrica del material depende de la dirección de la magnetización del medio que se mueve por debajo
- Se hace pasar una corriente a través del sensor MR y los cambios de resistencia se detectan como señales de voltaje
- Provee mayores densidades de grabación y velocidades de operación que el mecanismo anterior

- Organización (cont.)

- CAV (Constant Angular Velocity): el disco gira a velocidad constante
- La cabeza lectora/grabadora puede operar a la misma tasa de transferencia
- Los bits exteriores giran a mayor velocidad que los interiores (velocidad lineal variable)
- Para compensar, los bits exteriores están más espaciados entre sí
- Ventaja: se puede referenciar a cada bloque de información a través de pista/sector
- Desventaja: no se aprovecha el máximo de densidad (bits por pulgada lineal) de la superficie del disco

○ Grabación multizona

- La superficie del disco se divide en zonas concéntricas (por lo general 16)
- La cantidad de bits por pista dentro de una zona es constante
- Las zonas exteriores contienen más bits por pulgada (más sectores) que las zonas interiores

- Desventajas: mayor complejidad en la circuitería para trabajar con tiempos de lectura/escritura diferentes según la zona (la longitud de los bits varía)

- Características físicas

○ Movimiento de la cabeza

- Fija:** había una cabeza lectora/grabadora por pista (muy costosos, no se usan más) Ej. IBM 2305
- Móvil:** hay una única cabeza lectora/grabadora por superficie del plato. Se mueve por todas las pistas y está montada en un brazo

○ Portabilidad

- Discos no removibles:** disco rígido (se monta en un disk drive)
- Removibles:** se puede sacar y poner en la unidad (Ej. floppy disk)

○ Lados

- Un solo lado:** solo es usable una cara
- Dos lados:** el recubrimiento magnético está en ambas caras

○ Platos

- Un solo plato**
- Múltiples platos:** varios discos en un mismo disk drive
 - Cilindro: conjunto de pistas que están a la misma distancia relativa del centro en los platos de un disk drive

○ Mecanismo de la cabeza

- Contacto:** toma contacto con la superficie del disco (Ej. floppy)
- Espacio fijo:** se ubica a una posición fija por encima del disco
- Espacio aerodinámico (flotante):** se ubica flotante por sobre el disco gracias a la presión de aire que genera la rotación del disco

- Parámetros de performance

- ### ○ Desempeño del disco:
- deseada
- depende del computador, sistema operativo, módulo de E/S y controlador de disco
 - Tiempo de seek:** tiempo necesario para mover la cabeza lectora/grabadora a la pista deseada

- ### ○ Tiempo de demora rotacional o latencia:
- deseada
- tiempo de espera hasta que el sector deseado pasa por la cabeza lectora/grabadora

- ### ○ Tiempo de acceso:
- deseada
- tiempo necesario para estar en posición para escribir o leer
 - Tiempo acceso = tiempo de seek promedio + tiempo de latencia promedio**

- ### ○ Tiempo de transferencia:
- deseada
- tiempo necesario para transferir la información al disco

$$\text{② } T = b/rN \text{ donde } T = \text{tiempo de transf.; } b = \text{bytes a transf.; } r = \text{velocidad de rotación en revoluciones por segundo; } N = \text{bytes por pista}$$

- ### ○ Tiempo total lectura/escritura

$$\text{② } T = T_{\text{seek}} + 1 / 2r + b / rN$$

- r: velocidad de rotación en revoluciones por segundo
- b: bytes a transferir
- N: bytes por pista

Tiempo de transferencia = bytes transferir / velocidad de rotación en rev seg * N/pista

○ Ejemplo:

- #### ② Comparación de tiempos de lectura

- Disco: tiempo seek promedio = 4ms; 15000 RPM; sectores de 512 bytes; 500 sectores por pista
- Archivo: 2500 sectores (1,28 MB)

- #### ② Organización secuencial

- #### ② Organización aleatoria

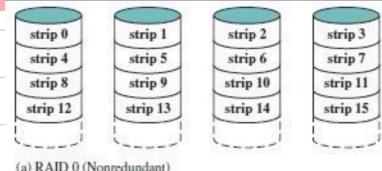
RAID (Redundant Array of Independent Disks) -> forma de agrupar distintos discos para generar

- Hay distintas maneras de organizar la información y agregarle confiabilidad a los datos
 - RAID es un estandar que consiste en 7 niveles (0 a 6). Pueden implementarse combinaciones de niveles (ej. RAID 0+1, RAID 5+0, etc.)
 - Es un conjunto de discos que son vistos por el sistema operativo como una única unidad lógica
 - Los datos se distribuyen en los discos del vector en un esquema llamado "striping" -> se graban en partecitas
 - Se usa capacidad redundante para guardar información de ante fallas (a excepción de RAID 0)

- Niveles

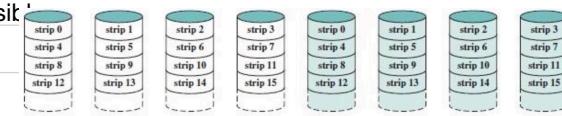
○ Nivel 0 (Striping)

- No incluye redundancia
- Uso : edición de video, apps con amplio ancho de banda
- Se requieren N discos
- Se distribuyen los datos en el vector de discos en strips (pueden ser sectores, bloques u otra unidad)
- Ventajas:
 - Simplicidad
 - Performance
- Desventaja:
 - Riesgo ante fallos, no hay recuperación posible



(a) RAID 0 (Nonredundant)

○ Nivel 1 (Espejado)

- ?
 Redundancia por espejado de datos


(b) RAID 1 (Mirrored)

usos: apps q requieren alta disponibilidad de datos

- ?
 Ventajas:

- Un pedido de lectura puede resolverse por cualquiera de los dos discos
- La escritura se hace en forma independiente en cada disco y no se penaliza
- Simple recuperación ante fallas porque lo tengo 2 veces
- Alta disponibilidad de datos

- ?
 Desventajas:

- Costo

○ Nivel 2 (Redundancia por código de Hamming)

- ?
 Strips pequeños (un byte o palabra)

- ?
 Se calcula redundancia por código autocorrector (ej. Hamming)

- ?
 Se requieren N + m discos

- ?
 Se graban bits de paridad en discos separados

- ?
 Se leen/escriben todos los discos en paralelo

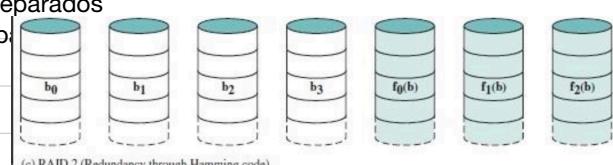
- ?
 No existe uso comercial

- ?
 Ventajas:

- Disponibilidad de datos

- ?
 Desventaja:

- Costos por método de redundancia



(c) RAID 2 (Redundancy through Hamming code)

○ Nivel 3 (Paridad por intercalamiento de bits)

- ?
 Solo se usa un disco de paridad

- ?
 Se requieren N+1 discos

usos: producción de video, apps q requieren alto rendimiento

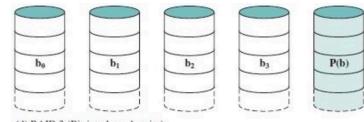
- ?
 La paridad se calcula mediante un bit a través del conjunto individual de bits de la misma posición de todos los discos

- ?
 Se leen/escriben todos los discos en paralelo, en forma sincronizada

- Cálculo sencillo de paridad
- No hay impacto significativo de performance ante fallas

? Desventajas:

- Controlador complejo



○ Nivel 4 (Paridad por intercalamiento de bloques)

- ?** Se accede en forma independiente a cada disco
- ?** Se requieren N+1 discos
- ?** Se puede dar servicio a pedidos de E/S en paralelo
- ?** Se usan strips grandes.

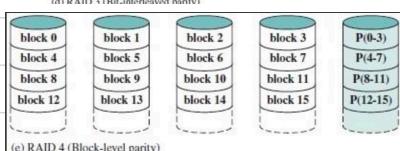
- ?** Los bits de paridad se calculan igual que en RAID 3 y se guarda un strip de paridad
- ?** No hay uso comercial

? Ventajas:

- Altas tasas de lectura

? Desventaja:

- Dos lecturas y dos escrituras en caso de update de datos
- Cuello de botella por disco de paridad -> Cuando se superponen actualizaciones en los discos en el disco de paridad se forma un cuello de botella.



○ Nivel 5 (Paridad por intercalamiento distribuido de bloques)

- ?** Se accede en forma independiente a cada disco

- ?** Se requieren N+1 discos

usos: servers , nivel RAID + versátil

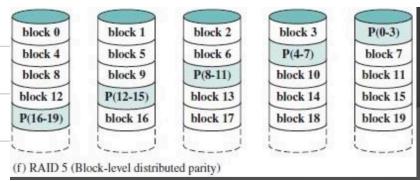
- ?** Los strips de paridad se distribuyen en todos los discos

? Ventajas:

- Resuelve el cuello de botella del nivel 4 -> disco de paridad distribuido por ende al actualizar discos pueden no coincidir al mismo tiempo

? Desventajas:

- Controlador complejo



○ Nivel 6 (Doble paridad por intercalamiento distribuido de bloques)

- ?** Se accede en forma independiente a cada disco **?** Se requieren N+2 discos

- ?** Se usan dos algoritmos de control de paridad

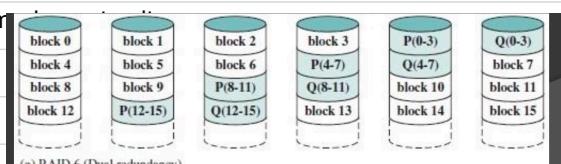
usos: para aplicaciones de misiones críticas

? Ventajas:

- Provee disponibilidad de datos extremo

? Desventaja:

- Controlador complejo
- Costos por doble paridad



Medios ópticos

- CD (Compact Disc)

- Estándar internacional (1980) entre Philips y Sony (Red book)
- 120 mm de ancho x 1,2 mm de espesor
- CLV (Constant Linear Velocity) -> misma velo en cualquier parte del disco
- Pista única en forma de espiral
- Sectores (2352 bytes)
- Master:
 - ?** Se graba con un laser infrarrojo de alto poder
 - ?** Se marcan huecos sobre un disco de vidrio
 - ?** Se hace un molde y se le inyecta policarbonato que sigue los patrones de los huecos
 - ?** Se pone luego una capa reflectiva de aluminio sobre el sustrato
 - ?** Encima se coloca una capa protectora de laca y una etiqueta

- Land: espacio de la superficie sin marcas
 - Bit en 1: transición entre pit-land o land-pit
 - Bit en 0: ausencia de transición en un tiempo determinado => CLV
- **CD-ROM (Compact Disc-Read Only Memory)**
 - Estándar internacional (1984) (Yellow book)
 - Encoding EFM (Eight to Fourteen Modulation)
 - Modos
 - Modo 1 (Datos): 2048 bytes de datos
 - Modo 2 (Audio): 2336 bytes => 74 minutos
- **CD-R (Compact Disc-Recordable)**
 - Estándar internacional (1989) (Orange book)
 - Se podía escribir solo una vez
 - En vez de hacer huecos físicos se usa una capa de tinta que inicialmente es transparente y deja pasar la luz laser
 - El laser quema la capa de tinta y crea un punto negro que ya no refleja la luz simulando un pit
 - CD-XA
- **CD-RW (Compact Disc-Rewriteable)**
 - Se usa un material que puede tener dos estados con reflectividades diferentes
 - Cristalino: refleja la luz
 - Amorfo: no refleja la luz
 - Para escribir se usa alto poder en el laser para pasar el material de estado cristalino a amorfo (representa un pit)
 - Para borrar se usa medio poder en el laser y se pasa el material de estado amorfo a cristalino nuevamente
 - Para leer se usa bajo poder en el laser
- **DVD (Digital Versatile Disk)**
 - Diseño general igual al de los CDs
 - Pensados para la industria de distribución de video
 - Mejoras con respecto al CD:
 - Pits más pequeños (0,4 µm versus 0,8 µm del CD)
 - Espiral más encimada (0,74 µm entre pistas versus 1,6 µm del CD)
 - Laser color rojo (a 0,65 µm versus 0,78 µm del CD)
 - Capacidad mejorada a 4,7 GB
 - Formatos definidos:
 - Un solo lado, una sola capa (4,7 GB)
 - Un solo lado, dos capas (8,5 GB)
 - Dos lados, una sola capa (9,4 GB)
 - Doble lado, dos capas (17 GB)
 - DVD-R: usa tecnología similar a CD-R para permitir hacer una grabación única
 - DVD-RW: usa tecnología similar a CD-RW para poder grabar/borrar varias veces
- **HD (High Definition)**
 - Se crearon dos estándares para poder distribuir video de alta definición
 - HD-DVD (High Definition – Digital Versatile Disk)
 - Blue-ray Disc (BD)
 - Blue-ray fue la tecnología que predominó
 - Se basa en un laser azul de 0,405 µm
 - La capacidad es de 25 GB para la versión de una sola capa y 50 GB para la de doble capa
 - BD-R: usa tecnología similar a CD-R para permitir hacer una grabación única

SSD

- SSD (Solid State Drive)
 - Definición : "Dispositivo de almacenamiento secundario hecho con componentes electrónicos de estado sólido (semiconductores)"
 - Historia
 - Basados en RAM (volátiles – energía auxiliar)
 - Texas memory: 16KB (1978)
 - Basados en flash (no volátiles)
 - M-Systems (1995) Tecnología actual
 - NAND Flash
 - Arquitectura
 - Controlador
 - Cache
 - Memorias NAND flash
 - Condensador
 - Comparación con discos magnéticos
 - Ventajas
 - Arranque más rápido
 - Gran velocidad de lectura y escritura
 - Baja latencia(tiempo necesario para ejecutar) de lectura y escritura
 - Menor consumo de energía
 - Menor producción de calor
 - Sin ruido
 - Mejor MTBF (tiempo medio entre fallas) + raro q se rompan algún disco
 - Mayor seguridad de datos
 - Rendimiento determinístico
 - Menor peso y tamaño
 - Mayor resistencia a golpes, caídas y vibraciones
 - Desventajas
 - Precio (\$/GB)
 - Menos recuperación ante fallos
 - Capacidad
 - Vida útil
 - Tecnologías SSD
 - PCIe / SATA (interface externa)
 - AHCI / NVME (interface de comunicación) M.2 / 2.5" SATA / mSATA (form factor)

Preguntas y respuestas de finales

Final 2/8/22

- 1) [1,5 ptos] ¿Cuál es la finalidad de la existencia de los números desnormalizados en el formato de punto flotante IEEE 754? Grafique todos los valores extremos del rango de números desnormalizados y dé sus configuraciones hexadecimales en el formato.

- Estos números tienen como exponente al cero¹ y el bit 1 que se hallaba implícito a la izquierda del punto binario es ahora un 0 implícito. Los números desnormalizados se distinguen de los normalizados porque estos últimos no permiten al cero como exponente.

- Sirven para disminuir el hueco entre el 0 y el número normalizado más pequeño, Permiten reducir paulatinamente la magnitud de un número hasta llegar a 0.

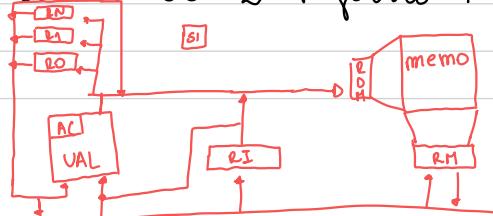
Desnormalizado ± 0 Cualquier patrón de bits $\neq 0$

El número desnormalizado más chico consiste de un 1 en la posición menos significativa de la fracción, y ceros en las posiciones restantes. En este caso el exponente es -126 y la fracción representa 2^{-23} , siendo entonces el valor del número $2^{-23} \times 2^{-126} = 2^{-149}$.

- 1) [1,5 ptos] Explique claramente por qué se dice que la UAL en SuperAbaus se utiliza tanto para la suma de datos como de direcciones. Justifíquelo con microinstrucciones y el gráfico de la máquina.

* En Superabaus la UAL se utiliza tanto para la suma de datos como de direcciones ya que en el R1 luego de ejecutar 1 instrucción se actualiza la próxima dirección de instrucción a ejecutar lo cual se realiza incrementando el R0 para obtener la próxima dirección. A su vez tb existe el modo de direccionamiento Base + desplazamiento, el cual se utilizará para la suma de 2 registros lo q hace es tomar el 1º registro y sumarle el contenido de la celda de memoria cuya dirección es la suma del 2º registro + el offset

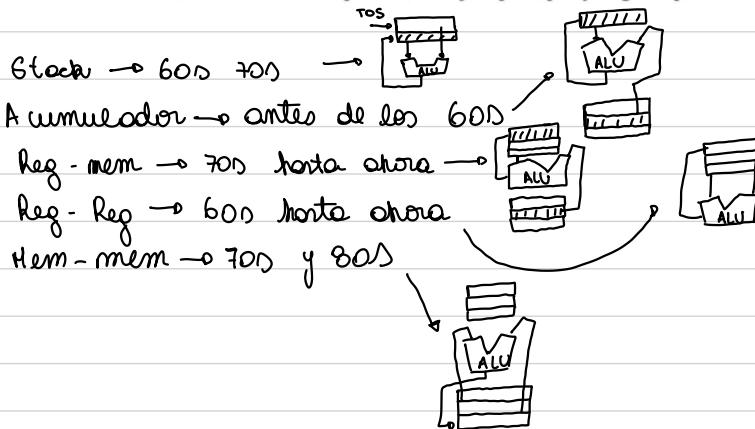
AC $\leftarrow (R3) + 20$
RDM $\leftarrow (AC)$
RM $\leftarrow ((RDM))$
AC $\leftarrow (RM) + (R5)$
R5 $\leftarrow (AC)$



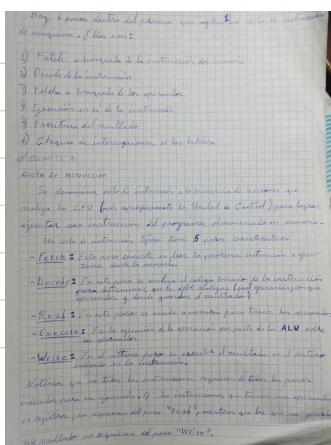
- 4) [1 pto] ¿Por qué se dice que los formatos de instrucción de la arquitectura Intel x86 son variables? De algún ejemplo que justifique su respuesta.

* En Intel x86 se dice q los formatos de instrucción de la arquitectura no son variables ya que dependiendo de como el programador escriba las instrucciones va a terminar definiendo el tamaño de la instrucción ej: en intel se puede reservar memoria de a 1 byte(db) de a 2 (dw), de a 4(dd), de a 8(dq).

- 4) [1 pto] Indique como se puede clasificar el repertorio de instrucciones de una arquitectura de computadores de acuerdo a la ubicación de los operandos. Ejemplifique y/o grafique cada una.

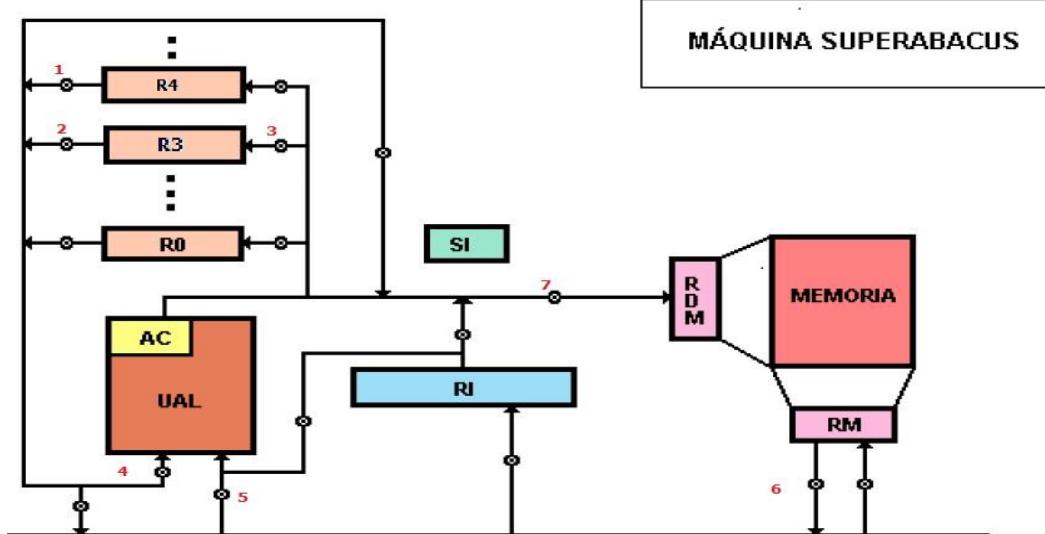


- 6) [1,5 ptos] Explique claramente que es un ciclo de instrucción en un procesador, indique qué etapas contempla y que ocurre durante la ejecución de cada una de ellas.



75.03 organización del computador

- [1,5 puntos] Indique cuales son las microinstrucciones necesarias para ejecutar la instrucción SUMAR 3,20(4) en una maquina Superabacus, siendo 3 y 4 registros de uso general y 20 un offset en base 10. Se pide además graficar en el esquema el flujo de apertura de compuertas usadas en la fase de ejecución de dicha instrucción.



AC	\leftarrow	(R4)	1,4
AC	\leftarrow	(AC)+20	
RDM	\leftarrow	(AC)	7
RM	\leftarrow	((RDM))	
AC	\leftarrow	(RM)	6,5
AC	\leftarrow	(AC)+(R3)	2,4
R3	\leftarrow	(AC)	3

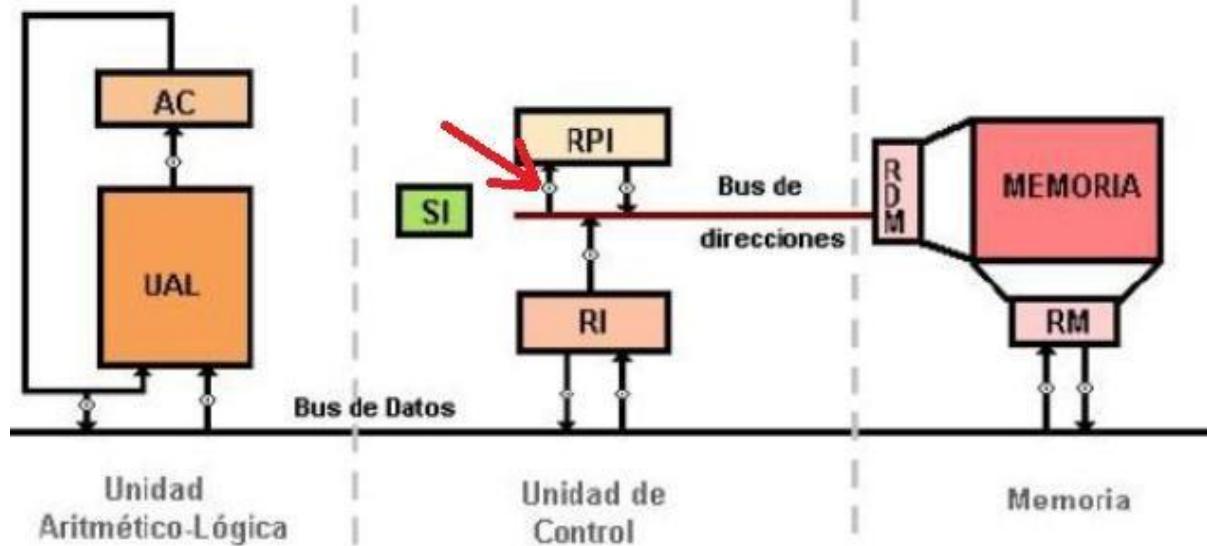
- [1,5 puntos] Indique cuales son las microinstrucciones necesarias para ejecutar la instrucción SUMAR 3,100 en una maquina SuperAbacus, siendo 3 un registro de uso general y 100 un offset en base 10. Se pide además graficar en el esquema, el flujo de apertura de compuertas usadas en la fase de ejecución de dicha instrucción.

Figura de Abacus anterior

AC	\leftarrow	(R3)	1,4
AC	\leftarrow	(AC)+100	
R3	\leftarrow	(AC)	3

- [1,5 ptos] Indique gráficamente en el esquema de la máquina Abacus cuál es la compuerta que permite que se cumpla el principio de

ruptura de secuencia de Von Neumann. De un ejemplo de una instrucción en donde se aplique este principio.



La instrucción es la de salto:

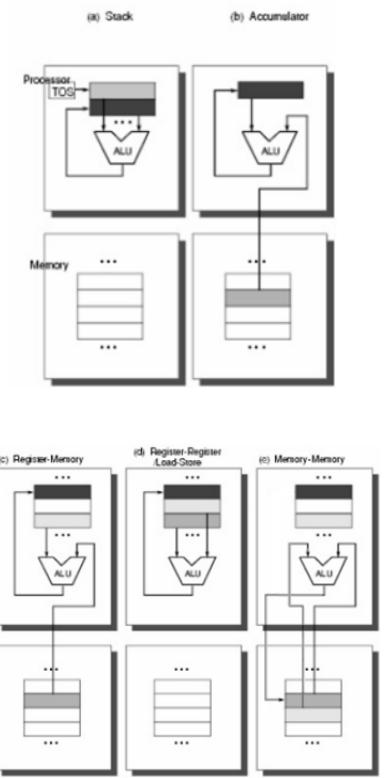
$$RPI \leftarrow (OP)$$

Se debe “saltar” a la dirección indicada en la instrucción. La dirección de bifurcación debe ser transferida al RPI (Para buscar la próxima instrucción). Esto rompería con la secuencialidad del programa y saltaría a otro sector de código. Esto cumpliría con el principio de Von Neumann de ruptura de secuencia, la cual es la clave del poder decisorio de los programas.

- [1 punto] Indique como se puede clasificar el repertorio de instrucciones de una arquitectura de computadores de acuerdo con la ubicación de los operandos. Ejemplifíque y/o grafique cada uno.**

$$C = A + B$$

- Stack (Pila)
 - PUSH A
 - PUSH B
 - ADD (Toma operandos de la pila)
 - POP C
- Acumulador
 - Load A (en el acumulador)
 - ADD B (Acumulador implícito)
 - STORE C (Guarda el contenido del acumulador en memoria)
- Registro-Memoria
 - Load R1, A (en registro R1)
 - ADD R3, R1, B (Suma B con R1 y guarda en R3)
 - STORE R3, C (en memoria)
- Registro-Registro
 - Load R1, A
 - ADD R2, B
 - STORE R3, R1, R2
 - Store R3, C
- Memoria
 - MOVE C, A
 - ADD C, B



- [1,5 puntos] Explique claramente cuáles son las características del modo de acceso asociativo y que en tipo de memoria está presente.

- Tipo de acceso aleatorio por comparación de patrón de bits.
- La palabra se busca por una porción de su contenido en vez de por su dirección.
- Cada posición de memoria tiene un mecanismo de direccionamiento propio.
- Tiempo de acceso constante, independiente de las secuencias de accesos anteriores o su ubicación.

Este modo de acceso está presente en las memorias caché. Estas memorias guardan los últimos bloques usados por el CPU. Este le pide una palabra y esta es buscada en la caché, si llegara a estar devuelve la palabra indicada. Si no se busca el bloque, en el que esta palabra se encuentra, en memoria principal. Luego se guarda el bloque en la caché y se entrega la palabra deseada por el procesador.

- [1,5 puntos] En la arquitectura de discos RAID de nivel 3: ¿Qué ocurre si un disco queda inhabilitado? ¿Cómo se puede recuperar la información perdida?

Este nivel de la arquitectura de discos RAID consiste en N discos de datos más uno de paridad.

La paridad se calcula con un sencillo bit de paridad para conjuntos de bits individuales en la misma posición en todos los discos de datos.

La reconstrucción de datos es bastante sencilla considerando 5 discos, de los que de X_1 a X_3 contienen datos y el X_4 es el de paridad. La paridad del *iésimo* bit se calcula de la siguiente forma:

$$X_4(i) = X_3(i) + X_2(i) + X_1(i) + X_0(i)$$

Siendo + la función or exclusivo

Si poniendo que fallo la unidad $X_1(i)$. Si sumamos $X_1(i) + X_1(i)$ a los dos miembros de la ecuación, nos quedaría:

$$X_1(i) = X_4(i) + X_3(i) + X_2(i) + X_0(i)$$

- [1,5 puntos] ¿Para qué existen las interrupciones? ¿Qué es lo que tratan de mejorar?

Las interrupciones son un mecanismo por el cual un módulo puede interrumpir al procesador en lo que estaba haciendo para que pase a hacer otra cosa. Existen para mejorar la performance global del procesador y la eficiencia en el procesamiento. Si no existieran, el CPU se quedaría esperando y recién ahí seguiría ejecutando. También sirven para avisarle que puede retornar un proceso al CPU. Como ejemplo concreto, las interrupciones de E/S logran maximizar el uso del procesador

- [1,5 puntos] Explique cuáles son los modos de direccionamiento presentes en la maquina SuperAbacus. De ejemplos de cada uno de ellos.

Los modos de direccionamiento presentes en una máquina SuperAbacus son:

- Inmediato

SUMAR	R4		100
-------	----	--	-----

- Registro Directo

SUMAR	R4	R5	
-------	----	----	--

- Registro indirecto

SUMAR	R4	R5	
-------	----	----	--

- Base + Desplazamiento

SUMAR	R4	R5	100
-------	----	----	-----

- **[1 punto] Enumere por lo menos 4 elementos presentes en la arquitectura de programación (ISA) de un computador. De ejemplos de dichos elementos en alguna de las arquitecturas vistas en clase.**

- Repertorio de instrucciones (instrucciones de maquina).
- Registros (Cantidad, tipo, tamaño).
- Formatos de instrucción.
- Modos de direccionamiento.
- Memoria
 - o Espacio de direcciones.
 - o Tamaño de la celda.
 - o Formato de palabra.
- Tipos de datos.
- Tipos de operandos.
- Especificaciones de la operación de las instrucciones.
- Interrupciones.

- **[1,5 puntos] ¿Qué es un “page fault” y cuando ocurre?**

En el sistema de administración de memoria paginada por demanda, se van subiendo las páginas que se necesitan. Cuando el CPU pide una página que no está cargada en memoria principal se produce una interrupción llamada “Page Fault”. Esta interrumpe el proceso y provoca que el sistema valla a buscar la hoja deseada a memoria secundaria.

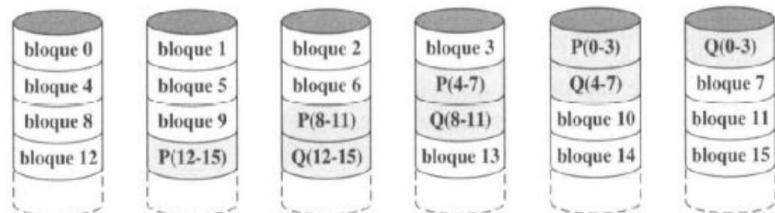
Un algoritmo de asignación ubica inicialmente algunas páginas del programa en la memoria. Por cada requerimiento de acceso a una página se consulta la tabla para saber en qué bloque esta. Si la página no está en memoria, se demanda la carga de esa página en memoria. Se detiene el procesamiento y genera automáticamente una interrupción, esta interrupción se conoce como interrupción de página (page fault). Es entonces el sistema operativo el que ejecuta la función de acceder al almacenamiento secundario para adquirir la página pedida.

- [1,5 puntos] ¿Cuáles son las ventajas y desventajas del nivel 6 de la arquitectura de discos RAID respecto al nivel 5? Grafique la distribución de la información en los discos de ambos niveles.**

RAID 6

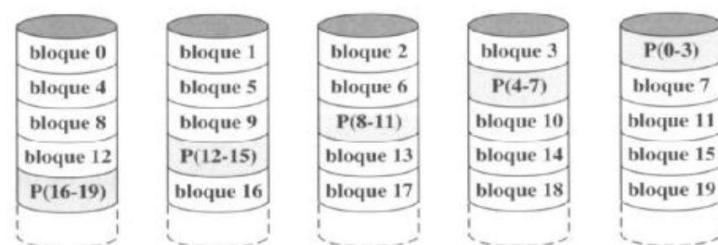
Ventajas:

- Proporciona una tolerancia a fallos mayor.
- Puede soportar la rotura de hasta 2 unidades simultáneamente.



Desventajas:

- Diseño del más complejo.
- Más costoso ya que se necesita un disco más (en nivel 5, n discos de datos +1 de redundancia; en nivel 6, n discos de datos +2 de redundancia).

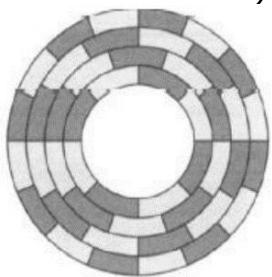


RAID 5

- [1,5 puntos] Explique claramente cuáles son las ventajas y desventajas de la organización tradicional de discos magnéticos versus la organización multi zona. Grafique ambas organizaciones.**

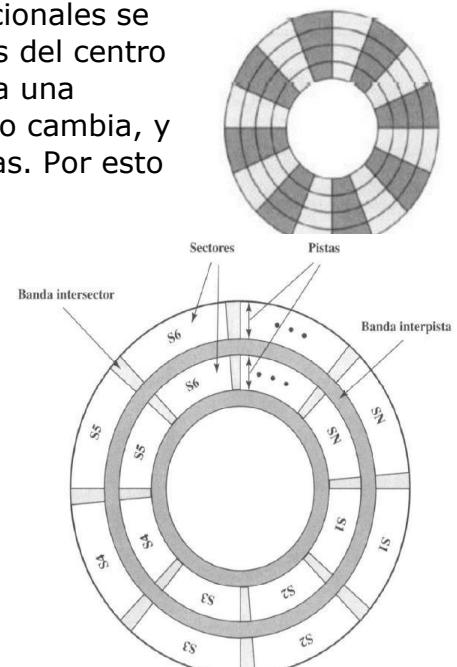
Ventajas:

Mayor capacidad de almacenamiento. En los discos tradicionales se tenían igual cantidad de bits en los sectores más alejados del centro y en los más cercanos. Esto era así debido a que se leía a una velocidad angular constante. En los discos multi zona esto cambia, y permite mayor densidad de bits en las zonas más alejadas. Por esto la capacidad de almacenamiento viene limitada por la máxima densidad de grabación que se puede llevar a cabo en la misma más interna (en los discos tradicionales).



Desventajas:

La mayor capacidad de almacenamiento viene acompañada de una circuitería más compleja y es más difícil situar a la cabeza en la zona que deseamos, ya que en el método tradicional esto se puede direccionar directamente con la pista y el sector.



- [1,5 puntos] Mencione al menos 3 modos de direccionamiento presentes en la arquitectura Intel x86 dando un ejemplo de uso de cada uno en una instrucción.

- Inmediato
 - o mov ax,8h
- Registro-Registro
 - o mov ax, bx
- Directo
 - o mov ax, [200h]
- Indirecto
 - o mov ax, [si]
 - o mov ax, [si + 100h]
- Base/Relativo
 - o mov ax, [bp]
 - o mov ax, [bp + si +8h]

- [1,5 puntos] Explique claramente los mecanismos para atender múltiples interrupciones. Explique las ventajas y desventajas de cada método.

Existen 2 mecanismos:

- Inhibición de interrupciones:

Deshabilita una interrupción mientras otra está siendo procesada. Si una interrupción ocurre en ese tiempo, queda pendiente para que el procesador la chequee. Entonces cuando un programa se ejecuta y aparece una interrupción, se deshabilitan las demás. Cuando la rutina de atención de interrupciones (RAI) se termina, las interrupciones se habilitan antes de ejecutar el programa y el procesador chequea si hubo otras interrupciones adicionales.

- o Ventaja: El manejo es simple porque se habilitan/deshabilitan las interrupciones en un estricto orden secuencial.
- o Desventaja: No toma en cuenta la prioridad relativa o las que son críticas.

- Manejo de prioridades

Permite definir prioridades a las interrupciones basándose en una tabla de prioridades

- o Ventaja: Soluciona el problema con el otro mecanismo de que una interrupción más crítica debe ser llamada antes que las otras.

- [1,5 puntos] Mencione al menos 4 ventajas de los discos SSD frente a los discos duros mecánicos.

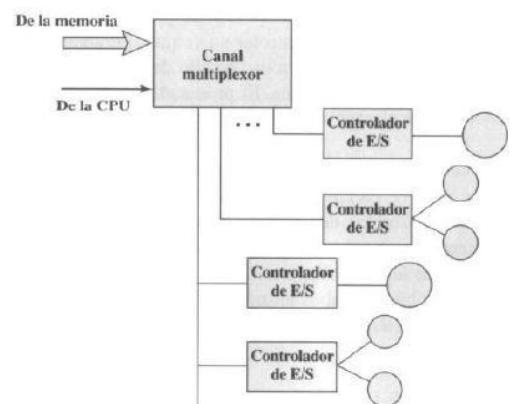
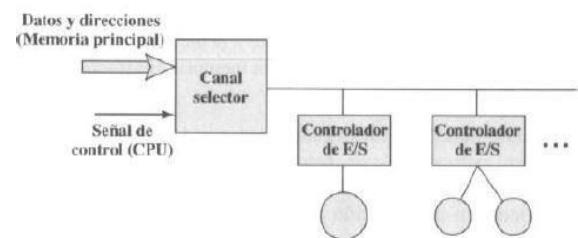
- Mayor velocidad de acceso a la información.
- Ruido (no hay).
- Calor (menos).
- Consumo de energía menor.
- Resistente a golpes.
- Seguridad al borrar datos.
- Rendimiento determinístico.

- [1,5 puntos] ¿Qué ventajas otorgan los canales de E/S? ¿Qué funciones cumplen el canal y que la CPU?**

El canal de E/S representa una aplicación del concepto DMA. Un canal de E/S puede ejecutar instrucciones de E/S (propias), lo que le confiere un control completo sobre las operaciones de E/S. En un computador de tales dispositivos, la CPU no ejecuta instrucciones de E/S. Dichas instrucciones se almacenan en memoria principal para ser ejecutadas por un procesador de uso específico contenido en el propio canal de E/S.

Son comunes dos tipos

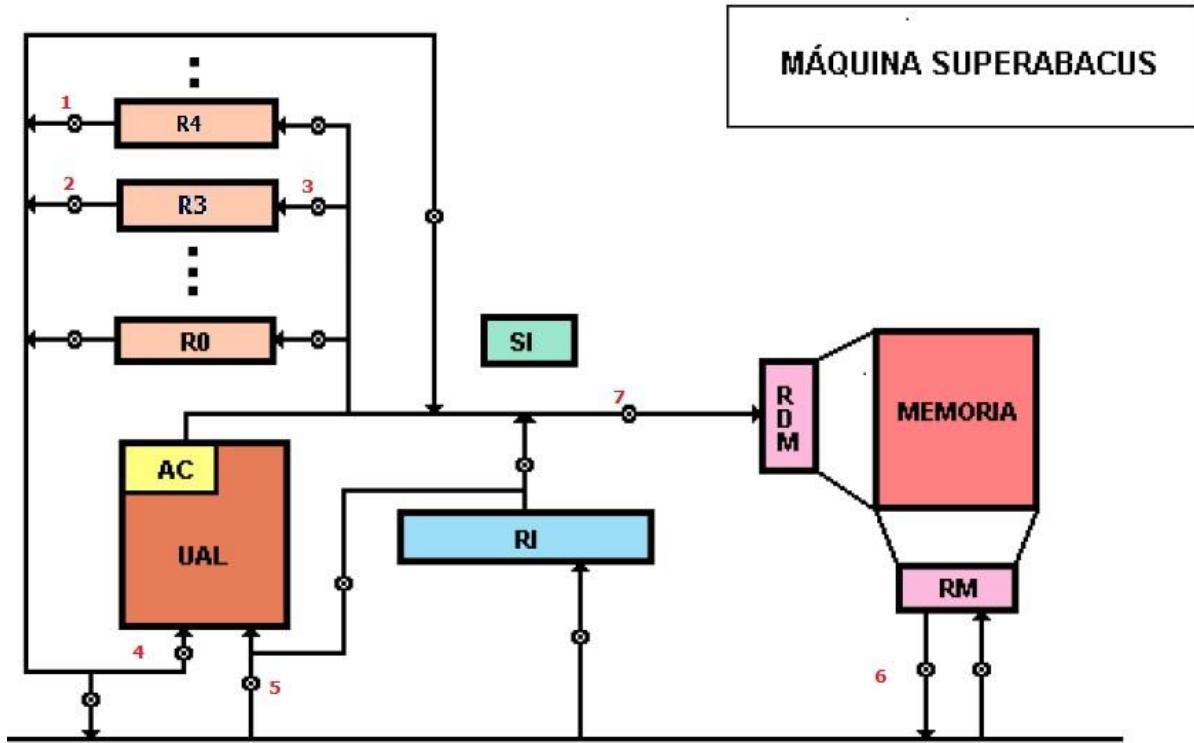
- Un canal selector
 - o Que controla varios dispositivos de velocidad elevada. El canal selecciona un dispositivo y efectúa la transferencia de datos. Cada dispositivo, o pequeño grupo de dispositivos, es manejado por un controlador.
- Canal multiplexor
 - o Puede manejar E/S de varios dispositivos al mismo tiempo. Para dispositivos de velocidad reducida.



- [1,5 puntos] Explique claramente al menos 3 funciones de los módulos de E/S.**

- Decodificación de ordenes
 - o El módulo de E/S acepta órdenes del procesador. Estas órdenes generalmente se envían utilizando líneas del bus de datos.
- Información de estado
 - o Puesto que los periféricos son lentos, es importante conocer el estado del módulo de E/S. También puede haber señales para informar ciertas situaciones de error.
- Detección de errores
 - o El módulo de E/S es responsable de detectar errores e informar al procesador. Una clase de errores son los defectos mecánicos y eléctricos en funcionamiento del dispositivo. Otra clase son los cambios accidentales en los bits al transmitirse desde el dispositivo al módulo de E/S.

- [1,5 puntos] Indique cuales son las microinstrucciones necesarias para ejecutar la instrucción SUMAR 3,4 en una maquina SuperAbacus, siendo 3 y 4 registros de uso general. Se pide además graficar en el esquema, el flujo de apertura de compuertas usadas en la fase de ejecución de dicha instrucción.**



AC	\leftarrow	(R4)	1,4
AC	\leftarrow	(AC)+R3	2,4
R3	\leftarrow	(AC)	3

- [1,5 puntos] Describa las características de los Métodos de acceso de unidades de datos directo y aleatorio.

Método de acceso directo (discos magnéticos)

- Dirección única para bloques o registros basada en su posición física.
- Tiempo de acceso variable.

Método de acceso Aleatorio (Memoria principal, algunas memorias caché)

- Cada oposición direccionable de memoria tiene un mecanismo de direccionamiento cableado físicamente.
- Tiempo de acceso constante, independiente de la secuencia de accesos anteriores.

- [1,5 puntos] ¿Qué es la codificación 8-14(EFM) y para qué se usa?

El protocolo 8-14 es una forma de codificar datos binarios, utilizada en medios ópticos. Se utiliza porque el sistema no es capaz de detectar dos 1 seguidos, entonces se intercalan 0. Para aplicarlo, se usa una tabla de doble entrada, donde a cada combinación de 8 bits le corresponde una de 14 bits sin unos contiguos.

- [1 punto] ¿Qué ventajas presenta el modo de direccionamiento por desplazamiento (relativo al PC/referencia al programa) frente al direccionamiento directo?

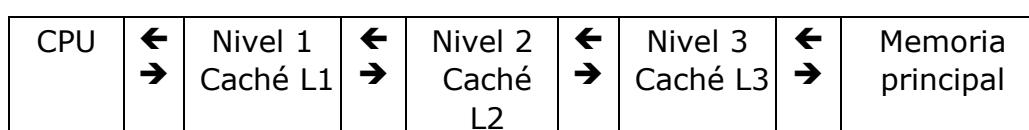
La ventaja del direccionamiento por referencia al programa y las demás formas de direccionamiento relativo es que permiten la reubicación de los programas en la memoria. En el caso particular del direccionamiento por referencia al programa, no

hay ningún cambio al reubicarlo ya que ese direccionamiento utiliza la dirección de la instrucción como referencia.

- [1,5 puntos] En un sistema de memoria, ¿Qué función cumple la memoria caché? ¿En qué principio se basa su efectividad? Grafique un ejemplo de la arquitectura de cache de 3 niveles.**

Su objetivo es lograr que la velocidad de la memoria sea lo más rápida posible. La caché contiene una copia de partes de memoria principal. Cuando el procesador intenta leer una palabra de memoria, se hace una comprobación para determinar si la palabra está en el caché. Si es así, se entrega dicha palabra. Si no, un bloque de memoria principal se transfiere a la caché y después la palabra es entregada al procesador.

Debido al fenómeno de localidad cuando un bloque de datos es capturado por la caché, es probable que se hagan referencias futuras a la misma posición de memoria o a otras palabras del mismo bloque.



- [1,5 puntos] Explique claramente cuáles son los eventos temporales presentes a la hora de almacenar o recuperar información en un disco magnético sectorizado. Especifique como haría el cálculo de lectura de un archivo con una distribución aleatoria de la información en el disco. Ejemplifíquelo de ser necesario.**

- Tiempo de seek (en un sistema de cabeza móvil): El tiempo que tarda la cabeza en posicionarse en la pista.
- Latencia rotacional: Una vez en la pistas, es el tiempo que tarda el disco en girar hasta el sector apropiado.
- Tiempo de acceso: La suma del tiempo de seek y la latencia rotacional.
- Tiempo de transferencia de datos: Lo que se tarda en hacer la operación de lectura/escritura.

$$T_{total} = T_{seek} + \frac{1}{2}R + \frac{b}{RN}$$

R = Velocidad de rotación en revoluciones por segundo.

b = Bytes a transferir.

N = Bytes por pista

- [1,5 puntos] ¿Cómo funciona el mecanismo de inhibición de interrupciones y para qué se usa? ¿Qué desventaja tiene?**

Deshabilita una interrupción mientras otra interrupción está siendo procesada. Si una interrupción ocurre en este tiempo, queda pendiente para que el procesador la chequee. Entonces cuando un programa se ejecuta y aparece una interrupción, se deshabilitan las demás. Cuando la rutina de atención de Interrupciones (RAI) se termina, las interrupciones se habilitan antes de ejecutar el programa y el procesador chequea si hubo otras interrupciones adicionales.

- Ventaja: el manejo es simple porque se habilitan/deshabilitan las interrupciones en un estricto orden secuencial
- Desventaja: no toma en cuenta la prioridad relativa o las que son críticas.

- [1,5 puntos] ¿Qué mecanismos provee el estándar IEEE 754 para el manejo de operaciones matemáticas con resultados indeterminados o indefinidos? De ejemplos de dichas operaciones e indique cual sería la configuración en el formato para representar dichos resultados.**

Valores no-numéricos: Denominados NaN (Not a number). Se identifican por un exponente con todos sus valores en 1, y un significando distinto de cero. Existen dos tipos de QNaN (Quiet NaN) y SNaN (Signalling NaN), que se distinguen dependiendo del valor 0/1 del bit más significativo del de la mantisa. QNaN tiene el primer bit en 1, y significa "Indeterminado". Resultado de todas aquellas operaciones aritméticas con resultados matemáticamente no definidos. SNaN tiene el primer bit en 0, y significa "operación no valida". Es la ejecución de una operación inválida. Ejemplos:

Operación	Resultado
Cualquier operación contra un NaN	NaN
+0 /+0	NaN
Infinito – Infinito	NaN
+Infinito / Infinito+-	NaN
+Infinito X 0	NaN

	Signo	Exponente en exceso	Mantisa
QNaN	0/1	11111111	1 0101010101001010101101
SNaN	0/1	11111111	0 1010101101101010101010

- [1,5 puntos] Nombre al menos tres causas por las cuales es necesaria la existencia de los módulos de E/S para la interconexión de periféricos con el resto del sistema.**

- Para liberar al procesador del trabajo de E/S.
- Para adaptar las velocidades de trabajo entre el CPU y los dispositivos.
- Para proporcionar una interface estándar contra dispositivos disimiles.

- [1 punto] Enuncie al menos 4 características de la arquitectura de procesadores CISC.**

- Muchas instrucciones y muy complejas.
- Muchos formatos de instrucciones.
- Muchas instrucciones para acceder a operandos en memoria.
- Muchos modos de direccionamiento.
- Muchos tipos.
- Pocos registros.

- [1,5 puntos] ¿Qué ventajas provee la administración de memoria paginada frente a otros mecanismos más sencillos? ¿Qué desventaja presenta frente a la administración paginada por demanda?**

Ventajas de administración de memoria paginada frente a otros mecanismos más sencillos:

- Minimiza la fragmentación interna (no del todo ya que se calcula que se pierde media página por programa).
- Permite que los bloques de memoria (frames) puedan almacenarse sin estar necesariamente contiguos. Esto evita tener que reordenar la memoria (compactamiento).

Desventaja de la administración de memoria paginada frente a la administración de memoria paginada por demanda.

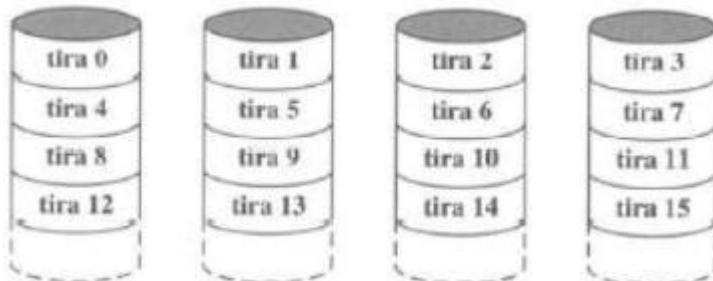
- Requiere que todas las páginas estén cargadas en la memoria.
- No permite ejecutar programas que no entran completamente en la memoria.

- [1,5 puntos] ¿Cuáles son las ventajas del nivel 1 de la arquitectura de discos RAID respecto al nivel 0? Grafique la distribución de la información en los discos en ambos niveles.

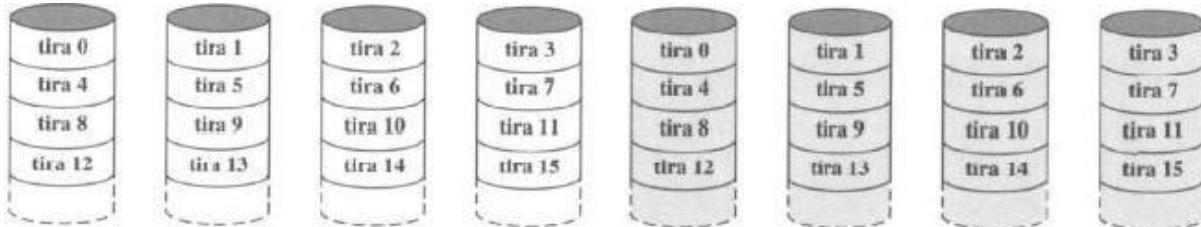
Ventajas

- Una petición de lectura puede ser servida por cualquiera de los discos que contienen los datos pedidos.
- La escritura se hace en forma independiente en cada disco.
- Alta disponibilidad de datos.
- La recuperación tras un fallo es muy sencilla, ya que están todos los datos por duplicado, lo que ofrece un 100% de redundancia.

RAID NIVEL 0



RAID NIVEL 1



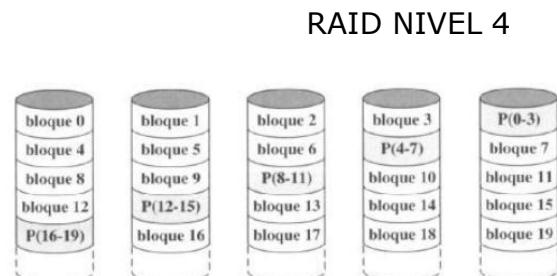
- [1 punto] Explique claramente qué es el fenómeno de "thrashing" y qué lo puede originar.

Si la CPU se ocupa de demasiados programas al mismo tiempo, como puede ocurrir cuando se usa paginación por demanda en un sistema con insuficiente memoria física (o algoritmos de juicio deficientes), no es raro que se carguen páginas en la memoria, se las suplante enseguida y luego se las vuelva cargar, y así sucesivamente. A esto se lo llama thrashing y provoca un gran deterioro en la performance.

- [1,5 puntos] ¿Cuáles son las ventajas del nivel 5 de la arquitectura de discos RAID con respecto al nivel 4? Grafique la distribución de la información en los discos en ambos niveles.**

Ventajas

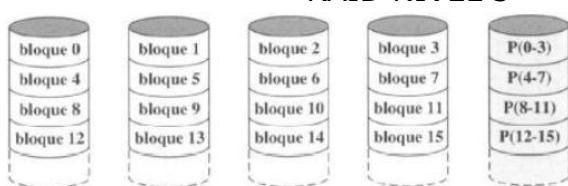
- Distribuye las tiras de paridad a lo largo de todos los discos. Lo cual evite un potencial cuello de botella de E/S encontrado en el RAID 4.



RAID NIVEL 4

Desventaja

- Controlador sumamente más complejo que el de su nivel inferior.



RAID NIVEL 5

- [1,5 ptos] ¿Qué mecanismos provee el estándar IEEE 754 para el manejo de números + - infinito? De ejemplos de dichas operaciones e indique cual sería la configuración en el formato para representar dichos resultados.**

Para los infinitos se ha convenido que cuando todos los bits del exponente están a 1 y todos los de la mantisa en 0, el valor es +/- infinito (según el bit de signo).

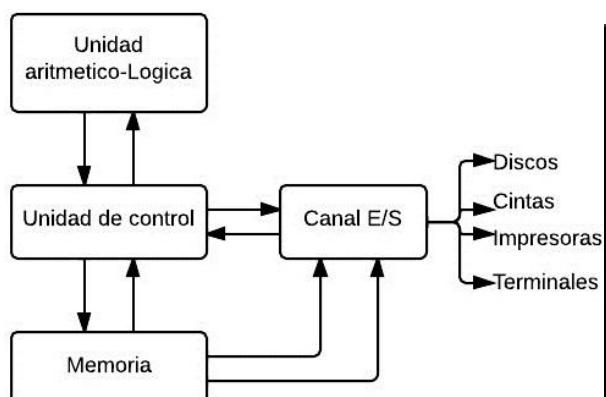
- [1,5 puntos] Hacer un cuadro comparativo de los componentes de almacenamiento externos, dar características, ventajas y desventajas de c/u**

Nombre	Características	Ventajas	Desventajas
Cintas magnéticas	-Acceso secuencial. - Está compuesta por 9 pistas, 8 bits de datos y 1 de paridad. - Generalmente usado para backup.	-Vida útil. -Capacidad de almacenamiento. -Bajo costo.	-Muy lento. -Baja tasa de transferencia de datos.
Discos magnéticos	-Acceso directo basado en su posición física. -Hay dos maneras en las que se disponen la información, la tradicional o la multizona.	-Mejor tasa de transferencia de datos que cintas magnéticas. -Mucha capacidad de almacenamiento (menor que cintas pero mayor que SSD).	-Genera mucho calor. -Ruidoso. -Fragmentación de datos en discos muy usados, esto puede generar un acceso casi aleatorio.

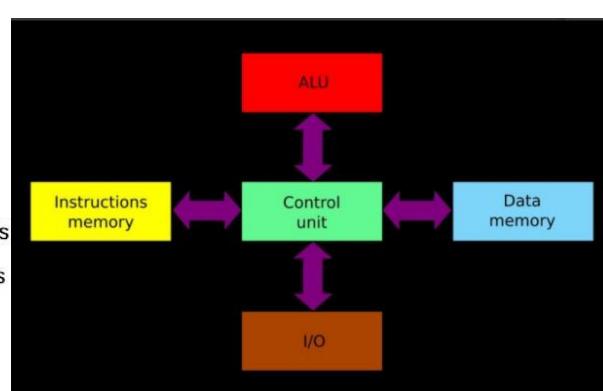
Medios ópticos	-Existe diferentes tipos (CD-Audio, CDROM,...,DVD, Blu-ray,...)	-Buena portabilidad. - Buena transferencia de datos.	-Poca vida útil. -Poca capacidad. -
Disco de estado sólido	-Posee un controlador, un buffer, memoria caché y un condensador.	-Silencioso -Rápida transferencia de datos. -No produce calor. -No tiene elementos mecánicos. -Mejor velocidad de acceso a la información -Consumo mínimo de energía. -Peso/Tamaño reducido	-Mayo precio por byte. -Menor capacidad que los discos magnéticos. -Menor vida útil que discos y cintas magnéticas. -No hay recuperación de la información ante fallos.

- [1,5 puntos] ¿Cuáles son las ventajas y desventajas de la arquitectura Harvard en relación con la arquitectura Von Neumann? Grafique ambas arquitecturas y justifique.

- Ventajas:
 - La división de la memoria, en una memoria de instrucciones y una memoria de datos
 - El procesador puede acceder a ambas memorias simultáneamente
 - Cada memoria y cada conexión de estas puede tomar distintas características. Tamaño de la memoria o la tecnología para su implementación
 - Mapa de direcciones y mapa de datos separados
- Desventajas:
 - La división de memorias solo funciona mejor en el caso particular, donde las frecuencias de lectura de instrucciones y de datos es aproximadamente la misma



Von Neuman



Harvard

- [1 punto] En la arquitectura ARM de 32 bits, ¿a qué se denomina ejecución condicional de una instrucción? De un ejemplo de su uso en assembler**

La instrucción es ejecutada sólo si el estado actual del flag del código de condición del procesador satisface la condición especificada en los bits b 31 -b 28 de la instrucción. Por lo tanto, las instrucciones cuya condición no se ve satisfecha en el flag de código de condición del procesador no se ejecutan. Una de las condiciones se utiliza para indicar que la instrucción siempre se ejecuta.

Esta característica elimina la necesidad de utilizar muchas bifurcaciones. El costo en tiempo de no ejecutar una instrucción condicional es frecuentemente menor que el uso de una bifurcación o llamado a una subrutina que, de otra manera, sería necesaria.

Para que una instrucción sea ejecutada condicionalmente se le agrega el sufijo con la condición apropiada. Por ejemplo, una instrucción de suma tiene la siguiente forma:

ADD r0, r1, r2

y para ejecutarla sólo si el flag cero está seteado:

ADDEQ r0, r1, r2

- [1,5 puntos] Explique claramente como funciona el Linking estático. Ejemplifíque y grafique dicho funcionamiento**

Cada módulo objeto compilado o ensamblado es creado con referencias relativos al inicio del módulo. Donde se combinan todos los módulos objeto en un único load module reubicable con todas las referencias relativas al load module.

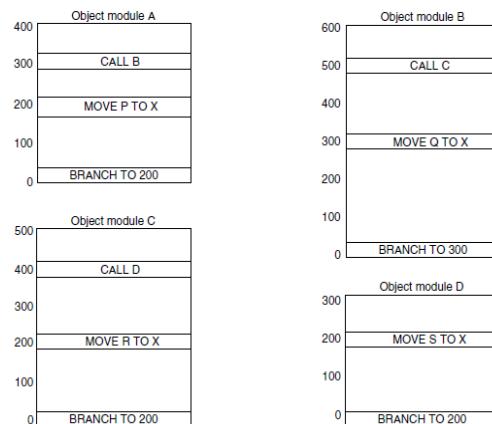


Figure 7-13. Each module has its own address space, starting at 0.

Module	Length	Starting address
A	400	100
B	600	500
C	500	1100
D	300	1600

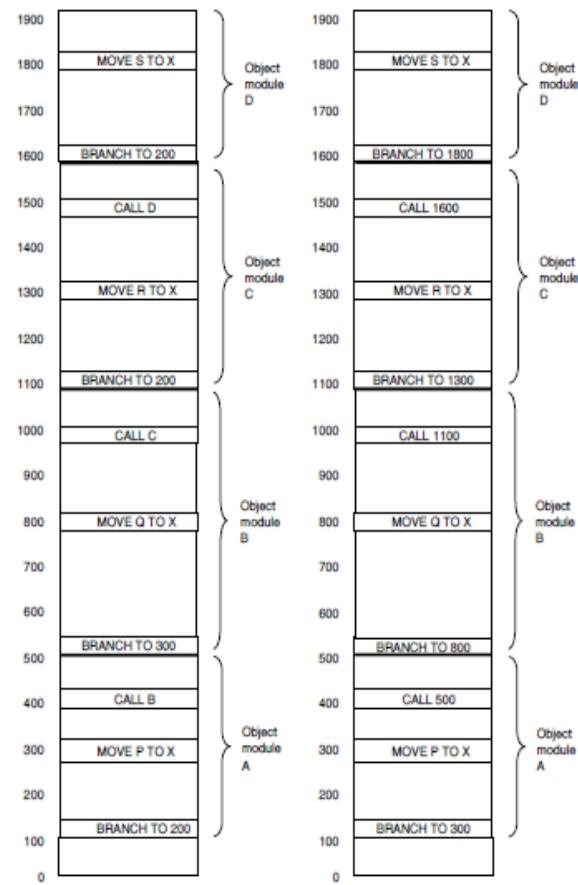


Figure 7-14. (a) The object modules of Fig. 7-13 after being positioned in the binary image but before being relocated and linked. (b) The same object modules after linking and after relocation has been performed.

- [1,5 puntos] Identifique y explique cuáles son las principales desventajas del medio de almacenamiento en cinta. ¿Cuáles son sus aplicaciones actuales? ¿Qué ventaja comparativa tiene con respecto al resto de los medios de almacenamiento secundario?

Cintas magnéticas son usadas actualmente para hacer backup y de archivo de información.

Ventajas:

1. Vida útil, pueden llegar a durar hasta 30 años
2. Posee grandes tamaños de almacenamientos de hasta PB
3. Bajo costo por byte: Con respecto a los demás medios de almacenamientos, estos se pueden producir a menor costo dado que el material usado es más barato y no requiere de componentes eléctricos, si lo comparamos con los medios ópticos, estos tienen poco espacio de almacenamiento

Desventajas:

- a Muy lento: Acceso secuencial a la información: si estoy en el registro 1 y quiero llegar al N tengo que "leer" los N-1 del medio. Si quiero leer un registro anterior tengo que rebobinar y volver a buscar el registro
- b Baja tasa de transferencia de datos. Por tener un grabado secuencial

- [1,5 puntos] Grafique el esquema general de un archivo de código objeto e identifique y explique cada una de sus secciones, indicando para que se usan**

- Identificación: nombre del módulo, longitudes de las partes del módulo
- Tabla de punto de entrada: lista de símbolos que pueden ser referenciados desde otros módulos
- Tabla de referencias externas: lista de símbolos usados en el módulo, pero definidos fuera de él y sus referencias en el código
- Código ensamblado y constantes
- Diccionario de reubicabilidad: lista de direcciones a ser reubicadas
- Fin de módulo

End of module
Relocation dictionary
Machine instructions and constants
External reference table
Entry point table
Identification

- [1 punto] Indique como se puede clasificar el repertorio de instrucciones de una arquitectura de computadores de acuerdo con el número de direcciones. Ejemplifíque y/o grafique cada uno.**

El conjunto de

- 0 direcciones (Stack)
Ejemplo: add $TOS \leftarrow TOS + Next$
- 1 dirección (Acumulador)
Ejemplo: add A $AC \leftarrow AC + Mem[A]$
- 2 direcciones (Reg-Mem/Reg-Reg/Mem-Mem)
Ejemplo: add R1, A $R1 \leftarrow R1 + Mem[A]$
- 3 direcciones (Reg/Mem)
Ejemplo: add R1, R2, R3 $R1 \leftarrow R2 + R3$

- [1 punto] Explique claramente y ejemplifíque al menos cuatro modos de direccionamiento presentes en la arquitectura ARM 32 bits**

Nombre	Nombre Alternativo	Ejemplos
Registro a registro	Registro directo	mov r0, r1
Absoluto	Directo	ldr r0, mem
Literal	Inmediato	mov r0, #15 add r1, r2, #12
Indexado, base	Registro indirecto	ldr r0, [r1]
Pre-Indexado, base con desplazamiento	Registro indirecto con offset	ldr r0, [r1, #4]
Pre-indexado, autoindexado	Registro indirecto con pre-incremento	ldr r0, [r1, #4]!
Post-indexado, autoindexado	Registro indirecto con post-incremento	ldr r0, [r1], #4
Doble registro indirecto	Registro indirecto indexado	ldr r0, [r1, r2]
Doble registro indirecto escalado	Registro indirecto indexado escalado	ldr r0, [r1, r2, lsl #2]
Relativo al PC		ldr r0, [PC, #offset]

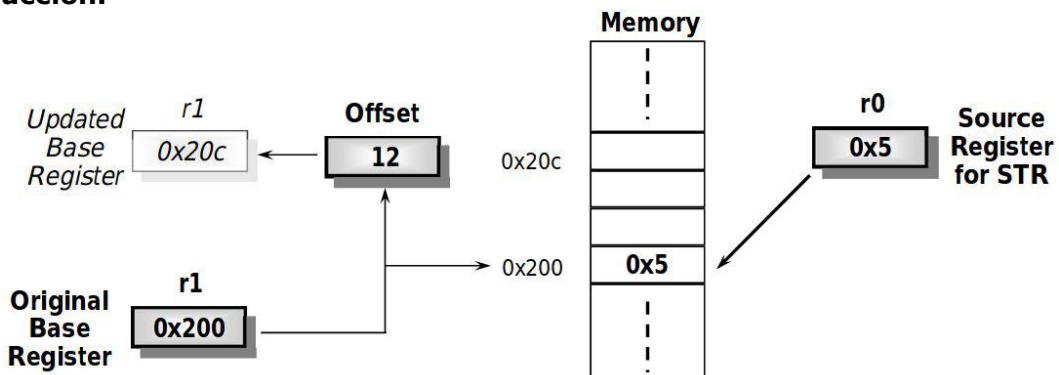
- [1,5 puntos] Indique claramente que es Linking dinámico en tiempo de ejecución y cuáles son las diferencias frente al Linking dinámico**

1. Se pospone el lindeo hasta el tiempo de ejecución
2. Se mantienen las referencias a módulos externos en el programa cargado
3. Cuando efectivamente se invoca al módulo externo, el sistema operativo lo busca, lo carga y lindea al módulo llamador.

Ventajas

- i No ocupo memoria hasta que la necesito (ej. Bibliotecas DLL de Windows)

- [1 pto] Explique claramente que es y cómo funciona el modo de direccionamiento post-indexado autoindexado (registro indirecto con post-incremento) en la arquitectura ARM de 32 bits. De un ejemplo concreto con una instrucción.**



La dirección efectiva del operando es el contenido de Rn. El desplazamiento se agrega a esta dirección y el resultado se escribe de nuevo en Rn.

[Rn], #offset

DE=[Rn]

Rn ← [Rn]+offset

[Rn], ±Rm, shift

DE=[Rn]

Rn ← [Rn] ± [Rm] shifteado

Primero guardo después actualizo!

- [1 pto] Explique claramente que es y cómo funciona el barrel shifter en la arquitectura ARM de 32 bits. De ejemplos concretos con instrucciones assembler**

ARM no tiene instrucciones de shift. En su lugar tiene un barrel shifter que provee un mecanismo que lleva a cabo shifts como parte de otras instrucciones.

Cuando se especifica que el segundo operando es un registro shifteado, la operación del Barrel Shifter es controlada por el campo Shift en la instrucción.

Este campo indica el tipo de shift a realizar. La cantidad de bits a shiftear puede estar contenida en un campo inmediato o en el byte inferior de otro registro (que no sea el R15)

1. Shift a Izquierda (LSL): Shift a la izquierda según la cantidad especificada (multiplica por potencias de 2). Por ejemplo:

LSL # 5 @ multiplica por 32

2. Shift Lógico a Derecha (LSR): Shift a la derecha según la cantidad especificada (divide por potencia de 2). Por ejemplo:

LSR # 5 @ divide por 32

3. Shift Aritmético a Derecha (ASR): Shift a la derecha según la cantidad especificada (divide por potencia de 2) preservando el bit de signo. Por ejemplo:

ASR # 5 @ divide por 32

Preguntas con ejercicios de codificación

- [2 puntos] Codificar un programa en assembler ARM de 32 bits que imprima tres cadenas de caracteres (definidas en el propio programa) por la salida estándar, haciendo uso de una subrutina interna

```
.equ SWI_PrStr, 0x69
.equ SWI_Exit, 0x11

.data
first_string:
    .asciz "Hola\n"
second_string:
    .asciz "Chau\n"
third_string:
    .asciz "como estas?\n"

.text
.global _start
_start:
    ldr r3, =first_string
    bl print_r3
    ldr r3, =second_string
    bl print_r3
    ldr r3, =third_string
    bl print_r3
    b fin

print_r3:
    stmdfd sp!, {r0,r1,lr}
    mov r0, #1
    mov r1, r3
    swi SWI_Print_String
    ldmfd sp!, {r0,r1,pc}

fin:
    swi SWI_Exit
.end
```

- [2 puntos] Codificar un programa en assembler ARM de 32 bits que recorra un vector de enteros y genere un nuevo vector formado por elementos que resultan de sumar(restar) pares de elementos del vector original. Ej. Vector original {1,2,5,6}, vector nuevo {3,11}

```
.equ SWI_Print_Int, 0x6B
.equ SWI_Exit, 0x11
.equ SWI_Print_Str, 0x69
.equ Stdout, 1

.data
array_origen:
```

```

.word 1,2,5,6
array_destino:
.word 0, 0, 0, 0
array_length:
.word 4
eol:
.asciz "\n"

.text
.global _start
_start:

ldr r0, =array_origen
ldr r1, =array_destino
ldr r2, =array_length
ldr r2, [r2]

loop_suma:
ldr r4, [r0]
add r0, r0, #4
sub r2, r2, #1
ldr r5, [r0]
add r6, r4, r5 @Si es resta se cambia la operacion por subs
str r6, [r1]
add r0, r0, #4
add r1, r1, #4
sub r2, r2, #1
cmp r2, #0
bne loop_suma

ldr r2, =array_destino
ldr r3, =array_length
ldr r3, [r3]

loop_mostrar:
cmp r3, #0
beq exit
ldr r0, =Stdout
ldr r1, [r2]
swi SWI_Print_Int
ldr r1, =eol
swi SWI_Print_Str
add r2, r2, #4
sub r3, r3, #1
b loop_mostrar

exit:
swi SWI_Exit
.end

```

- [2 puntos] Codificar un programa en assembler ARM de 32 bits que recorra un vector de enteros y los imprima por la salida estándar agregando la leyenda “PAR” a continuación de todos aquellos que así lo sean

```

.equ SWI_Print_Int, 0x6B
.equ SWI_Exit, 0x11
.equ SWI_Print_Str, 0x69
.equ Stdout, 1

.data
array_origen:
.word 1,2,5,6
array_length:
.word 4
eol:
.asciz "\n"
par:
.asciz " PAR"

.text
.global _start
_start:
ldr r0, =array_origen
ldr r2, =array_length
ldr r2, [r2]
loop:
ldr r4, [r0]
bl imprimir
add r0, r0, #4
subs r2, r2, #1
cmp r2, #0
bne loop
b exit
imprimir:
stmfd sp!, {r0,r1,lr}
ldr r0, =Stdout
mov r1, r4
swi SWI_Print_Int
and r5,r4,#1      @Hago and en el último bit
cmp r5, #0        @comparo para saber si es par o no
bne impSig
ldr r1, =par
swi SWI_Print_Str
impSig:
ldr r1, =eol
swi SWI_Print_Str
ldmfd sp!, {r0,r1,pc}
exit:
swi SWI_Exit
.end

```

- [2 puntos] Codificar un programa en Assembler ARM de 32 bits que lea desde un archivo números enteros e imprima por la salida estándar la productoria de aquellos números que sean positivos

```

.equ SWI_Open_File, 0x66
.equ SWI_Read_Int, 0x6C
.equ SWI_Print_Int, 0x6B
.equ SWI_Close_File, 0x68
.equ SWI_Exit, 0x11
.equ SWI_Print_Char, 0x00
.equ SWI_Print_Str, 0x69
.equ Stdout, 1
.data

filename:
.asciz "enteros.txt"

eol:
.asciz "\n"
.align

InFileHandle:
.word 0
.text
.global _start

_start: @abrir archivo
ldr r0, =filename          @ nombre de archivo de entrada
mov r1, #0                  @ modo: entrada
swi SWI_Open_File           @ abre archivo
bcs InFileError            @ chequear si hubo error
ldr r1,=InFileHandle        @ cargar dirección donde almacenar el handler
str r0, [r1]                 @ almacenar handler
mov r4, #1                  @acumulador de productoria

read_loop: @leer entero de archivo
ldr r0, =InFileHandle
ldr r0, [r0]
swi SWI_Read_Int
bcs EofReached
mov r2, r0                  @ el entero está ahora en r0
@Compruebo si es positivo
mov r3, #0
subs r3, r3, r2             @Si la resta es negativa, r2>0
bmi productoria
b read_loop

productoria:
mul r4, r4, r2
b read_loop

InFileError:
EofReached:
ldr r0, =Stdout
mov r1, r4
swi SWI_Print_Int
swi SWI_Exit
.end

```

ARM → Temeplate

@ Constantes

.equ SWI_EXIT, 0x11

@ sección de datos

.data

message:

.asciz "

.word 0

@ Sección de código

.text

.global _start

_start:

log
hay
q
hacer

fin:

SWI_SWI_EXIT

.end

SWI → abrir archivo → 0x66
Cerra " → 0x68
leer Reg " → 0x6C
leer String " → 0x6A
exit → 0x11
Print int → 0x6B
" Ch → 0x00
" Str → 0x69

ro = nomarchivo
r1 = modo
ro = filehandle
ro = int
ro = file handle / byte
r1 = dire derribo
r2 = MAX bytes a atmice
ro = donde
r1 = int
ro = filehandle / stdout
r1 = string

ASM EJ1holamundo.s X ASM EJ2cadenascaracteres.s ASM EJ3calculosAritmeticos.s ASM holamundo.s ASM EJ2cadenascaracteres.s X ASM EJ3calculosAritmeticos.s

```
C: > Users > paula > OneDrive > Documentos > EJ1holamundo.s
1      @ constantes
2      .equ SWI_Print_String, 0x02
3      .equ SWI_Exit, 0x11
4
5      @ sección de datos - datos que el programa usará
6      .data
7      mensaje:
8      .asciz : string terminado en byte nulo
9      .asciz "Hola Mundo"
10
11     @ sección de código - porción ejecutable del código
12     .text
13
14     @ hace _start disponible al linker
15     @ esto es efectivamente parte de la definición del "main"
16     .global _start
17
18     _start:
19     @ carga el puntero al string en el registro r0
20     ldr r0, =mensaje
21
22     @ solicita a ARMSim que imprima el string
23     swi SWI_Print_String
24
25     @ solicita a ARMSim que salga del programa
26     swi SWI_Exit
27     .end

plamundo.s ASM EJ2cadenascaracteres.s ASM EJ3calculosAritmeticos.s ● ASM E4entradasalidaentero.s
ers > paula > OneDrive > Documentos > EJ3calculosAritmeticos.s
@ En este programa realizamos varias operaciones aritméticas sobre dos números.
@ No imprimimos los resultados, sin embargo podemos verlos en los registros de ARMSim
@ si nos movemos utilizando single-step a través del código
    @ constantes
    .equ SWI_Exit, 0x11
    @ sección de código - porción ejecutable del código
    .text
    @ hace _start disponible al linker
    @ esto es efectivamente parte de la definición del "main"
    .global _start
_start:
    @ carga los valores 5 y 2 en los registros r0 y r1
    mov r0, #5
    mov r1, #2

    @ (r2) = (r0) + (r1)
    add r2, r0, r1
    @ (r3) = (r0) - (r1)
    sub r3, r0, r1
    @ (r4) = (r0) * (r1)
    mul r4, r0, r1
    @ (r5) = (r0) AND (r1)
    and r5, r0, r1
    @ (r6) = (r0) OR (r1)
    orr r6, r0, r1
    @ (r7) = (r0) XOR (r1)
    eor r7, r0, r1
    @ (r8) = shift izquierdo de (r0) en (r1) bits
    mov r8, r0, LSL r1 @ LSL: logical shift left
    @ (r9) = shift derecho de (r0) en (r1) bits
    mov r9, r0, LSR r1 @ LSR: logical shift right
    @ (r10) = shift aritmético derecho de (r0) en (r1) bits
    mov r10, r0, ASR r1 @ ASR: arithmetic shift right

    @ solicita a ARMSim que salga del programa
    swi SWI_Exit
    .end

Jusers > paula > OneDrive > Documentos > EJ2cadenascaracteres.s
    .equ SWI_Print_String, 0x02
    .equ SWI_Exit, 0x11
    .data
first_string:
    .asciz "Hola\n"
second_string:
    .asciz "Chau\n"
    .text
    .global _start
_start:
    ldr r3, =first_string
    bl print_r3
    ldr r3, =second_string
    bl print_r3
    b fin

print_r3:
    stmdfd sp!, {r0,lr}
    mov r0, r3
    swi SWI_Print_String
    ldmfd sp!, {r0,pc}

fin:
    swi SWI_Exit
    .end

holamundo.s ASM EJ2cadenascaracteres.s ASM EJ3calculosAritmeticos.s ● ASM E4entradasalidaentero.s
ers > paula > OneDrive > Documentos > EJ4entradasalidaentero.s
    .equ SWI_Open_File, 0x66
    .equ SWI_Read_Int, 0x6C
    .equ SWI_Print_Int, 0x6B
    .equ SWI_Close_File, 0x68
    .equ SWI_Exit, 0x11
    @ sección de datos - datos que el programa usará
    .data
filename:
    .asciz : string terminado en byte nulo
    .asciz "entero.txt"
    @ sección de código - porción ejecutable del código
    .text
    @ hace _start disponible al linker
    @ esto es efectivamente parte de la definición del "main"
    .global _start
_start:
    @ carga el puntero al string en el registro r0
    ldr r0, =filename
    mov r1, #0          @ abrir para lectura
    swi SWI_Open_File @ abrir el archivo
    @ copia el manejador de archivo de r0 a r5
    mov r5, r0
    @ lee un entero desde el archivo
    @ PRECOND - r0: manejador del archivo
    @ POSCOND - r4: entero leído desde el archivo
    swi SWI_Read_Int
    @ mostrar el entero por pantalla
    @ PRECOND - r4: donde mostrar (1: stdout)
    @ PRECOND - r1: entero a mostrar
    mov r1, r4
    mov r6, #1
    swi SWI_Print_Int
    @ cerrar el archivo, which looks for
    @ PRECOND - r0: manejador del archivo
    mov r4, r5
    swi SWI_Close_File
    @ solicita a ARMSim que salga del programa
    swi SWI_Exit
    .end

```

```
holamundo.s El2Cadenascaracteres.s ElCalculosAritmeticos.s ElMenstradaslidaenteros.s ElNegarenteros.s volumundos El2Cadenascaracteres ElCalculosAritmeticos ElMenstradaslidaenteros ElNegarenteros Elmostrarcalu...  
users > paula > OneDrive > Documentos > ElNegarenteros.s  
.equ SWI_Open_File, 0x66  
.equ SWI_Read_Int, 0x6C  
.equ SWI_Close_Int, 0x6B  
.equ SWI_Close_File, 0x68  
.equ SWI_Exit, 0x11  
.equ SWI_Print_Char, 0x88  
.equ SWI_Print_Str, 0x80  
.equ Stdout, 1  
.data  
filename:  
.asciz "dos_enteros.txt"  
eol:  
.asciz "\n"  
.align  
InfileHandle:  
.word 0  

```