

Estructura de Computadores

Indice

Tema 1: [Introducción a la Asignatura Estructura de Computadores.](#)

Tema 2: [Arquitectura de Von Neumann.](#)

Tema 3: Representación Digital de la Información: [Los Datos: ASCII , N° Naturales, N° Enteros](#)

Tema 4: Operaciones Aritméticas y Lógicas: [Nº Naturales, Nº Enteros, Operaciones Lógicas](#)

Tema 5: Representación Digital de la Información: [las Instrucciones.](#)

Tema 6: Programación en Lenguaje Ensamblador (x86): [Construcciones básicas de los lenguajes de alto nivel.](#)

Ejercicios Teoría: [Ejercicios Propuestos Temas 1-6](#)

Programas ASM Propuestos: [Al finalizar el 1º Parcial](#)

Tema 3: [Representación de los Datos: Números Reales](#)

Tema 7: [Unidad Central de Proceso.](#)

Tema 8: [Mecanismos de Entrada/Salida](#)

Tema 9: [Unidad de Memoria.](#)

Tema 1 : Introducción a la Asignatura Estructura de Computadores.

Indice.

- Profesorado
- Organización Académica
 - Programa
 - Prácticas
 - Ejercicios
 - Evaluación
 - Metodología

Profesorado

- Profesor Cándido Aramburu Mayoz.
 - Doctor Ingeniero Telecomunicación (UPNA-Universidad Politécnica de Madrid)
 - Empresa Ikusi S.A. (Sistemas de Telemedida 1989)
 - Profesor Titular UPNA (Dpto Ingeniería Electrónica y Comunicaciones 2000)
- Profesor Teoría: Carlos Juan de Dios
- Profesor Prácticas: Andrés Garde
 - <https://www.etsit.upm.es/>
 - <https://www.velatia.com/es/empresas-que-forman-velatia/ikusi/>
 - <https://www.unavarra.es/eu/sites/Portada/home.html>

Organización Académica

Aulas y Horarios

- Aulas
 - Teoría: G91 → A019, G1 → A113, G2 → A122
 - Prácticas: P91 → A015, P1 → A305, P2 → E-ISM, P3 → E-ISM
 - E-ISM : Edificio “Las Encinas” , lado derecho entre la Biblioteca y el Rectorado) en el Sótano, Laboratorio de Informática “ISM”
- Horario
 - Teoría: G91(L-15:00), G1(X-17:00), G2(L-17:00)
 - Prácticas: P91(X-19:00), P1(J-17:00), P2(M-19:00), P3(M-17:00)

Tutorías

- Despacho: Edificio Los Tejos 2 Planta: Despacho 2028 (Prof. Candido Aramburu)
- Miaulario → correo interno
- [Tutorías](https://www.unavarra.es/pdi?uid=364&dato=tutorias) [<https://www.unavarra.es/pdi?uid=364&dato=tutorias>]
 - Lunes (10-13) y Miércoles (10-13)

Programa de la Asignatura

- [Ficha Web Upna](https://www.unavarra.es/ficha-asignaturaDOA?languageld=100000&codPlan=240&codAsig=240306&anio=2023) [<https://www.unavarra.es/ficha-asignaturaDOA?languageld=100000&codPlan=240&codAsig=240306&anio=2023>]
 - Programa en 3 partes
 - i. **C**
 - ii. **C**
 - iii. Otros:

Bibliografía

- Fundamentos de Electrónica Digital. Cecilio Blanco

Metodología

- Trabajo en clase: principalmente Ejercicios con su teoría asociada

Prácticas

- Tipo de prácticas:

Ejercicios

- x

Evaluación

- Sistema de Evaluación:

- 35% teoría , 35% prácticas, 15% programación en papel, 15% otros

Tema 2 : Arquitectura de Von Neumann

Indice

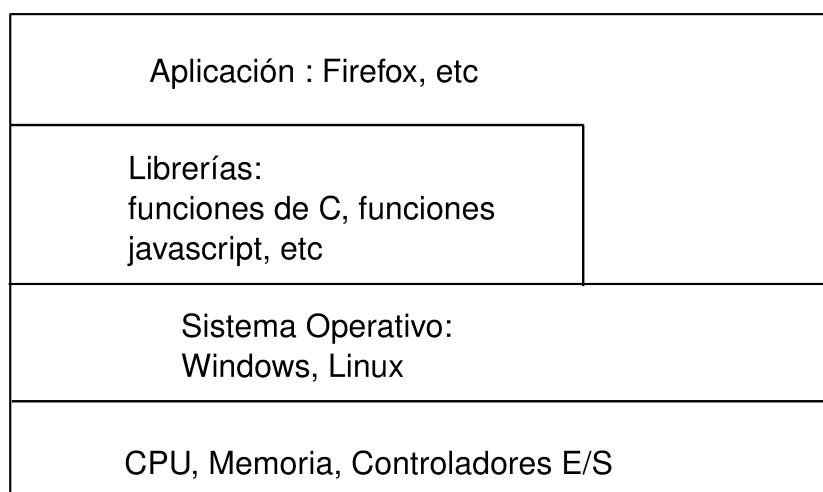
- P
- O

Qué conocemos de una computadora

- CPU? 3GHz? RAM? Memoria? 12GB? Disco Duro? Tarjeta gráfica? Pantalla? Ethernet? Placa base?
- Hardware, Software, Firmware? BIOS? Windows? Linux?
- Cuál es su arquitectura? Qué se entiende por arquitectura?

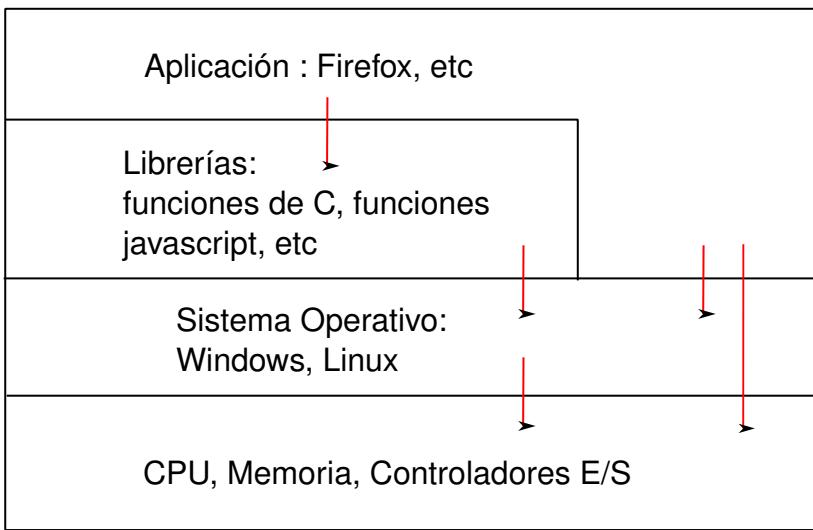
Organización de una computadora: niveles

- Los elementos básicos de una computadora desde el punto de vista del programador se pueden organizar por NIVELES:
 - del nivel más bajo y cercano a la máquina al nivel más alto y cercano al programador.



Organización de una computadora: llamadas

- Cada nivel llama a un nivel inferior



Organización de un automóvil: concepto de Abstracción

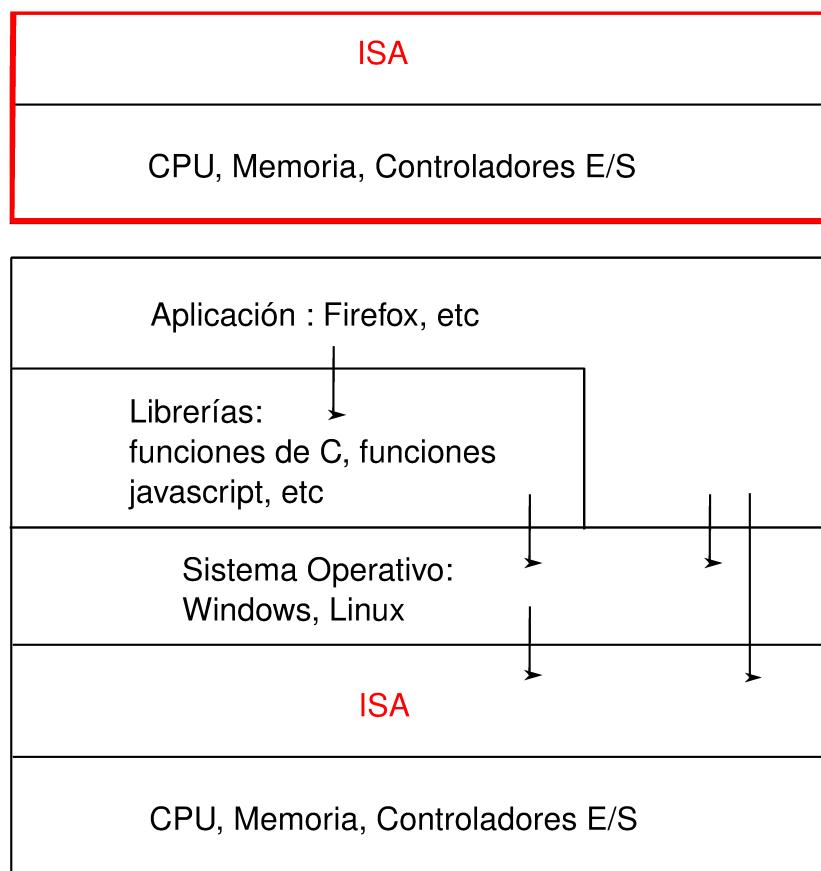
- Un símil sería la relación entre un conductor y el coche. El coche físico (motor,ventilador,ejes,inyector,válvulas,caja de cambios,etc...) se **ABSTRAE** y para el conductor un coche son unos pedales, una palanca de cambios y un volante. La Arquitectura del coche también se podría describir por niveles, del nivel más bajo al nivel más alto, al más abstracto.
- El conductor conduce (programa) una máquina ABSTRACTA: acelera,frena,cambia y gira. No tiene por qué saber que la máquina real, la física, lleva aceite, ni siquiera tiene que saber que son necesarias unas ruedas.
- Gracias a la abstracción las máquinas se acercan en su uso a los humanos.
- Es necesario saber el **QUE** hace una máquina y no el **COMO** lo hace.

Organización de una computadora: ISA

- El Hardware de la computadora tiene como función interpretar y ejecutar un conjunto de instrucciones con el objeto de procesar datos para obtener algún tipo de resultado.
 - lenguaje de la máquina : lenguaje binario
 - las instrucciones y los datos de la máquina están en lenguaje binario: 0,1
 - instrucciones máquina: P.ej sumar dos datos (números enteros) →
 - instrucción suma de 2 más 5 en código máquina: 0101010001000010101010101
 - misma instrucción máquina pero en lenguaje ensamblador (texto): add 2,5
 - La CPU mediante sus circuitos electrónicos digitales realiza la suma binaria de los dos datos binarios
- **I.S.A** : Instruction Set Architecture
 - Arquitectura del Conjunto de Instrucciones máquina de una computadora
 - Conjunto: sumar, restar, mover un dato, saltar a una instrucción de memoria
 - Arquitectura:
 - instrucciones: qué operaciones? ¿cuál es el tamaño de una instrucción? ¿cuantos operandos tiene? ¿cuál es su código binario? ¿cómo se hace referencia a un operando?
 - datos: tipo de datos: enteros? reales? su tamaño? su codificación?

Organización de una computadora: ISA

- El Hardware de la computadora tiene como nivel superior las instrucciones máquina que es capaz de ejecutar y los datos máquina que es capaz de ejecutar



Organización de una computadora: ISA

- Un programador de bajo nivel , pej los creadores de sistemas operativos, de compiladores, etc..., necesitan conocer la ISA de la máquina.
- La programación de un sistema operativo como Windows y Linux, en los lenguajes C y el lenguaje ensamblador, requiere conocer la arquitectura ISA.
- Los manuales ISA de un microprocesador tienen toda la información que necesita el programador, por lo tanto el programador de bajo nivel no necesita conocer todo el hardware interno de la CPU, de la memoria y de los controladores de entrada salida, sino únicamente la información disponible en el manual ISA.
- Para el programador la máquina es la ISA, el programador ve una máquina ABSTRACTA, ve la FUNCION de la máquina, QUÉ hace la máquina...y no COMO lo hace.
- En el laboratorio programaremos en lenguaje del nivel más bajo, es decir, en el lenguaje de la máquina pero no en binario sino en modo texto mediante el lenguaje ensamblador. Las operaciones de las instrucciones de la máquina en lenguaje ensamblador se expresan mediante mnemónicos como ADD, SUB, LOAD, MOV ... que son reducciones de palabras inglesas.
- Buscar en google los manuales isa de intel x86 de 64 bits que son ejecutados por los microprocesadores de intel: core i3, core i5, core i7, etc...

Una máquina abstracta y muchas reales: compatibilidad

- De la misma forma que cuan do aprendemos a conducir (pedales+cambio+volante) nos sirve para cualquier coche, la **arquitectura del conjunto de instrucciones** es la misma para múltiples

procesadores.

- La ISA amd64 ó x86-64 es la misma para todos los procesadores de Intel (core,xeon,etc...) o de AMD (Ryzen, etc). De esta forma un programa binario que se ejecute en un core-i7 también lo puede hacer en un Ryzen 9.HON

Programa en lenguaje texto: suma de los 5 primeros números enteros

- Algoritmo $\sum_{i=1}^5 i$
- Lenguaje de texto Python

```
sum(range(5,0,-1))
```

Programa en lenguaje texto: suma de los 5 primeros números enteros

```
/*
Programa: sum1toN.c
Descripción:
    1+2+3+4+...+N
*/
#include <stdlib.h>
int main ()
{
    short N=5,y;
    while(N>=0)
        y+=N--;
    exit(y);
}
```

```
/* Programa Fuente: sum1toN.java
compilación: javac sum1toN.java -> sum1toN.class
ejecución -> java -cp . sum1toN
*/
public class sum1toN {
// método main encapsulado en la clase class
    public static void main(String[] args) {
        System.out.println("Suma de Números enteros");
        int x=5, suma=0;
        while (x >= 0 ) {
            System.out.print( x );
            System.out.print(",");
            suma=suma+x;
            x--;
        }
    }
}
```

```

        System.out.print("\n");
        System.out.print("suma="+suma);
        System.out.print("\n");
    }
}

```

Módulo binario: en código binario

```

gcc -o sum1toN sum1toN.c
-> genera el módulo ejecutable sum1toN para la arquitectura x86-64 +
ls -l sum1toN -> 16696 bytes
file sum1toN
sum1toN: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV),
dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2,
BuildID[sha1]=8df3de7b9ba05ceb7861b39d703c8ef11161284a, for GNU/Linux 3.2.0,
not stripped

```

```

gcc -m32 sum1toM sum1toN.c
-> genera el módulo ejecutable sum1toN para la arquitectura x86-32
hexdump sum1toN
00000000 457f 464c 0102 0001 0000 0000 0000 0000
00000010 0003 003e 0001 0000 1060 0000 0000 0000
00000020 0040 0000 0000 0000 3978 0000 0000 0000
00000030 0000 0000 0040 0038

```

programa ejecutable en código **hexadecimal**, más compacto que el binario

1^a columna: direcciones, apuntan o hacen referencia a un byte

2^a columna:

--- 4 dígitos hexadecimales del contenido. Si cada dígito hexadecimal equivale a 4 bits, dos dígitos hex equivalen a 1 byte.

--- 457f: 2 bytes de código ejecutable

--- No podemos interpretar el código binario, pero el procesador x86-64 sí.

Interpretación del código binario (del programa sum1toN)

- Desensamblador: convierte el código binario en código texto ENSAMBLADOR
- objdump -d sum1toN

Disassembly of section .text:

```

0000000000001060 <_start>:
1060: f3 0f 1e fa          endbr64
1064: 31 ed                xor    %ebp,%ebp
1066: 49 89 d1              mov    %rdx,%r9
1069: 5e                   pop    %rsi
106a: 48 89 e2              mov    %rsp,%rdx

```

```

106d: 48 83 e4 f0        and    $0xfffffffffffffff0,%rsp
1071: 50                  push   %rax
1072: 54                  push   %rsp
1073: 4c 8d 05 86 01 00 00 lea    0x186(%rip),%r8
107a: 48 8d 0d 0f 01 00 00 lea    0x10f(%rip),%rcx
1081: 48 8d 3d c1 00 00 00 lea    0xc1(%rip),%rdi
1088: ff 15 52 2f 00 00    callq *0x2f52(%rip)
108e: f4                  hlt
108f: 90                  nop

```

.....

0000000000001149 <main>:

```

1149: f3 0f 1e fa        endbr64
114d: 55                  push   %rbp
114e: 48 89 e5            mov    %rsp,%rbp
1151: 48 83 ec 10        sub    $0x10,%rsp
1155: 66 c7 45 fc 05 00  movw   $0x5,-0x4(%rbp)
115b: eb 19                jmp   1176 <main+0x2d>
115d: 0f b7 45 fc        movzwl -0x4(%rbp),%eax
1161: 89 c2                mov    %eax,%edx
1163: 83 ea 01            sub    $0x1,%edx
1166: 66 89 55 fc        mov    %dx,-0x4(%rbp)
116a: 89 c2                mov    %eax,%edx
116c: 0f b7 45 fe        movzwl -0x2(%rbp),%eax
1170: 01 d0                add    %edx,%eax
1172: 66 89 45 fe        mov    %ax,-0x2(%rbp)
1176: 66 83 7d fc 00    cmpw   $0x0,-0x4(%rbp)
117b: 79 e0                jns   115d <main+0x14>
117d: 0f bf 45 fe        movswl -0x2(%rbp),%eax
1181: 89 c7                mov    %eax,%edi
1183: e8 c8 fe ff ff    callq  1050 <exit@plt>
1188: 0f 1f 84 00 00 00 00 nopl   0x0(%rax,%rax,1)
118f: 00

```

1^a columna: dirección de referencia a la instrucción

2^a columna: código máquina en hexadecimal

3^a columna: código de operación de la instrucción en **lenguaje ensamblador**

4^a columna: operandos de la instrucción en **lenguaje ensamblador**.

Lenguaje Ensamblador

- Es el lenguaje máquina traducido del binario a TEXTO para poder programar módulos fuente los programadores
- El formato de la instrucción ensamblador se compone principalmente de 2 campos:
 - campo del código de operación mediante MNEMONICOS como push(insertar),mov(mover),add(sumar),jmp(saltar),jne(saltar si not equal), etv

- campo de los operandos de la operación: son referencias a memoria donde está el operando ó se pone el propio operando

Institute Advanced Machine (IAS)

- [John von Neumann](https://es.wikipedia.org/wiki/M%C3%A1quina_de_von_Neumann) [https://es.wikipedia.org/wiki/M%C3%A1quina_de_von_Neumann]

 - Matemático húngaro 1903
 - Proyecto Manhattan → Bomba atómica
 - Concepto de Arquitectura von Neumann de un Computador

- [Arquitectura de von Neumann](https://es.wikipedia.org/wiki/Arquitectura_de_Von_Neumann) [https://es.wikipedia.org/wiki/Arquitectura_de_Von_Neumann]

 - [máquina ENIAC](https://es.wikipedia.org/wiki/ENIAC) [https://es.wikipedia.org/wiki/ENIAC]: programa cableado
 - [máquina EDVAC](https://es.wikipedia.org/wiki/EDVAC) [https://es.wikipedia.org/wiki/EDVAC]: programa almacenado → *unidad de memoria
 - arquitectura von neumann: unidad de procesamiento + unidad de memoria + unidad de entrada salida
 - [youtube EDVAC](https://www.youtube.com/watch?v=jhlSElbxO0E) [https://www.youtube.com/watch?v=jhlSElbxO0E]

Repertorio de la máquina IAS: tabla

Table 1. Instruction Set I

Instruction name	Instruction name	Op Code	Description	Register Transfer Language (RTL)
S(x)→A	LOAD M(X) C+	1	copy the number in Selectron location x into AC	AC ← M[x]
S(x)→A	LOAD -M(X) C-	2	same as #1 but copy the negative of the number	AC ← ~M[x]+1
S(x)→A	LOAD M(X) cM	3	same as #1 but copy the absolute value	AC ← M[x]
S(x)→A	LOAD - M(X) c-M	4	same as #1 but subtract the absolute value	AC ← AC- M[x]
S(x)→A	ADD M(X) h+	5	add the number in Selectron location x into AC	
S(x)→A	SUB M(X) h-	6	subtract the number in Selectron location x from AC	
S(X)→	ADD M(X) AhM	7	same as #5, but add the absolute value	
S(X)→	SUB M(X) Ah-M	8	same as #7, but subtract the absolute value	

Repertorio de la máquina IAS: tabla

Table 2. Instruction Set II

Instruction name	Instruction name	Op Code	Description	Register Transfer Language (RTL)
S(x)→R	LOAD MQ,M(X)	9	copy the number in Selectron location x into AR	
R→A	LOAD MQ	A	copy the number in AR to AC	
S(x)*R→A	MUL M(X)	B	Multiply the number in Selectron location x by the number in AR. Place the left half of the result in AC and the right half in AR.	
A/S(x)→R	DIV M(X)	C	Divide the number in AC by the number in Selectron location x. Place the quotient in AR and the remainder in AC.	
Cu→S(x)	JUMP M(X,0:19)	D	Continue execution at the left-hand instruction of the pair at Selectron location x	
Cu`→S(x)	JUMP M(X,20:39)	E	Continue execution at the right-hand instruction of the pair at Selectron location x	
Cc→S(x)	JUMP+ M(X,0:19)	F	If the number in AC is >= 0, continue as in #D. Otherwise, continue normally.	
Cc`→S(x)	JUMP+ M(X,20:39)	10	If the number in AC is >= 0, continue as in #E. Otherwise, continue normally.	

Repertorio de la máquina IAS: tabla

Table 3. Instruction Set III

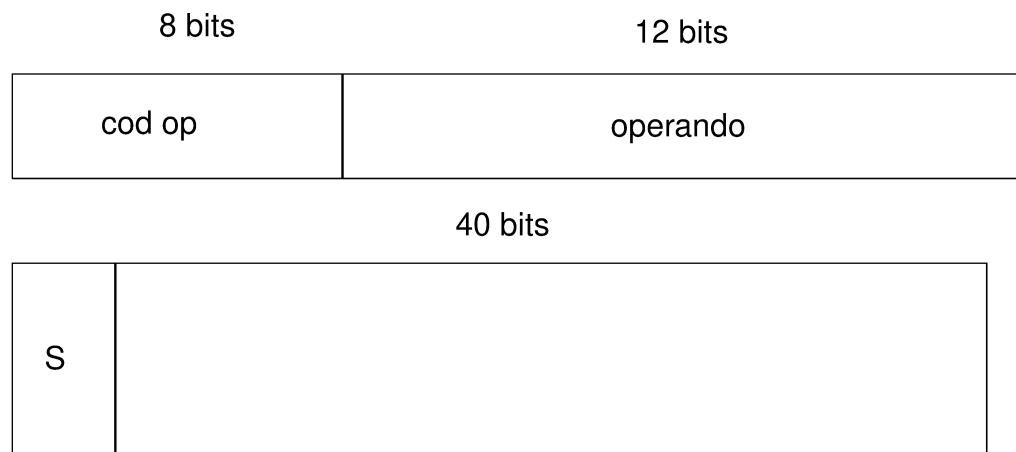
Instruction name	Instruction name	Op Code	Description	Register Transfer Language (RTL)
At→S(x)	STOR M(X)	11	Copy the number in AC to Selectron location x	
Ap→S(x)		12	Replace the right-hand 12 bits of the left-hand instruction at Selectron location x by the right-hand 12 bits of the AC	
Ap`→S(x)		13	Same as #12 but modifies the right-hand instruction	
L	LSH	14	Shift the number in AC to the left 1 bit (new bit on the right is 0)	
R	RSH	15	Shift the number in AC to the right 1 bit (leftmost bit is copied)	
halt		0	Halt the program (see paragraph 6.8.5 of the IAS r)	

Repertorio de la máquina IAS: explicación

- En la versión original no había código ensamblador, se programaba directamente en lenguaje máquina.
 - La 1^a columna: MNEMÓNICOS del lenguaje ensamblador del simulador académico IASSIM

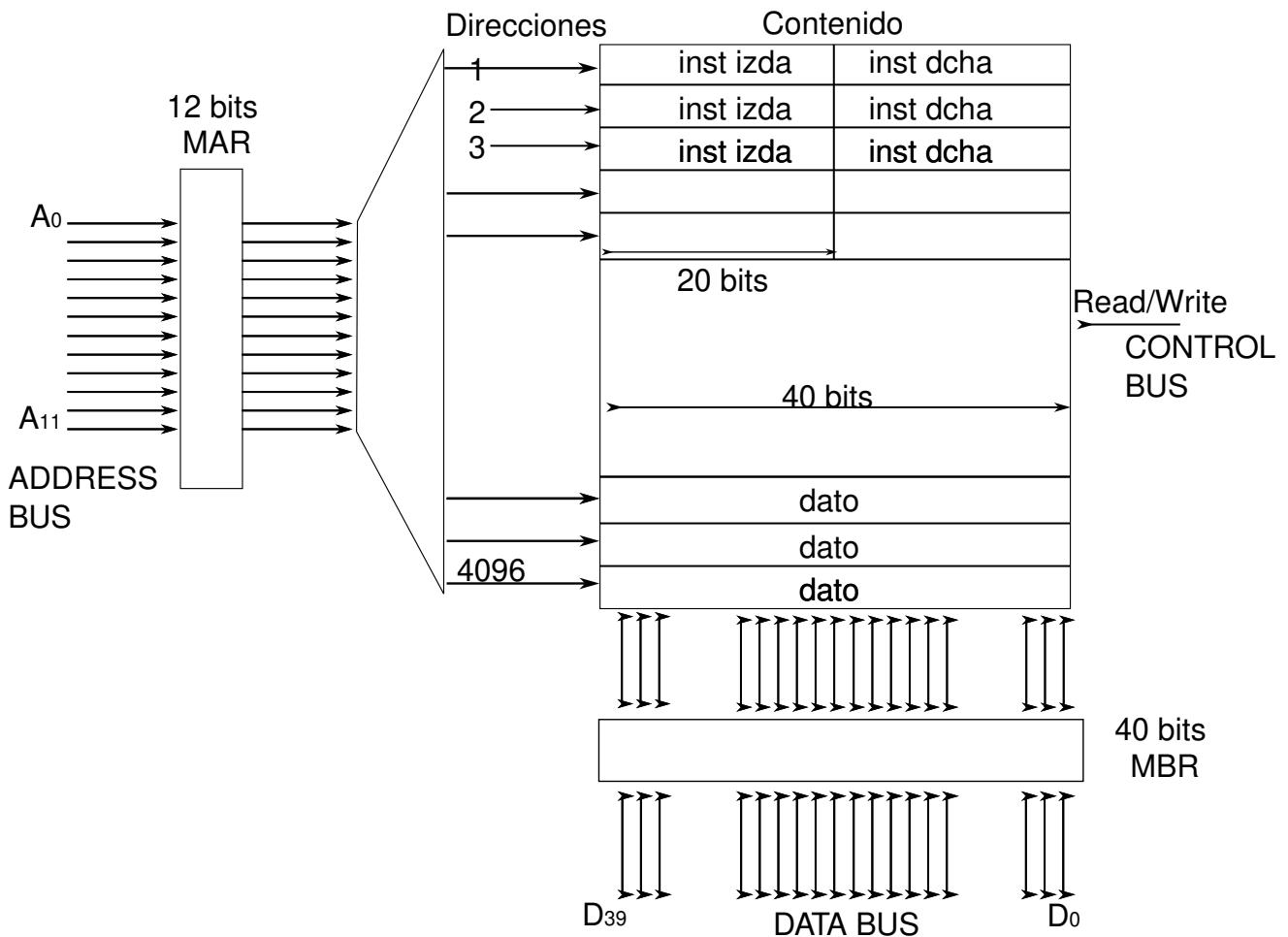
- En la 2^a columna, los **MNEMONICOS** (LOAD,ADD,SUB,etc) de las operaciones de las instrucciones se corresponden con los diseñados por el libro de texto de William Stalling.
- La 3^a columna describe la instrucción mediante el lenguaje de texto convencional
- La última columna describe la instrucción mediante un lenguaje de transferencia entre registros RTL

IAS de la máquina ISA



- Formato de datos
 - longitud de 40 bits
 - números enteros con signo: código complemento a 2
- Instrucciones: Sólo tiene 16 instrucciones por lo que el procesador es muy sencillo.
- Formato de las instrucciones
 - Tamaño fijo de 20 bit
 - la instrucción esta organizada en 2 campos: el campo de operaciones y el campo de operando
 - campo de operación: longitud de 8 bits → operaciones del tipo sumar,mover,saltar
 - campo de operando: longitud de 12 bits

Memoria de la máquina IAS



La memoria almacena el programa que ha de ejecutar la CPU

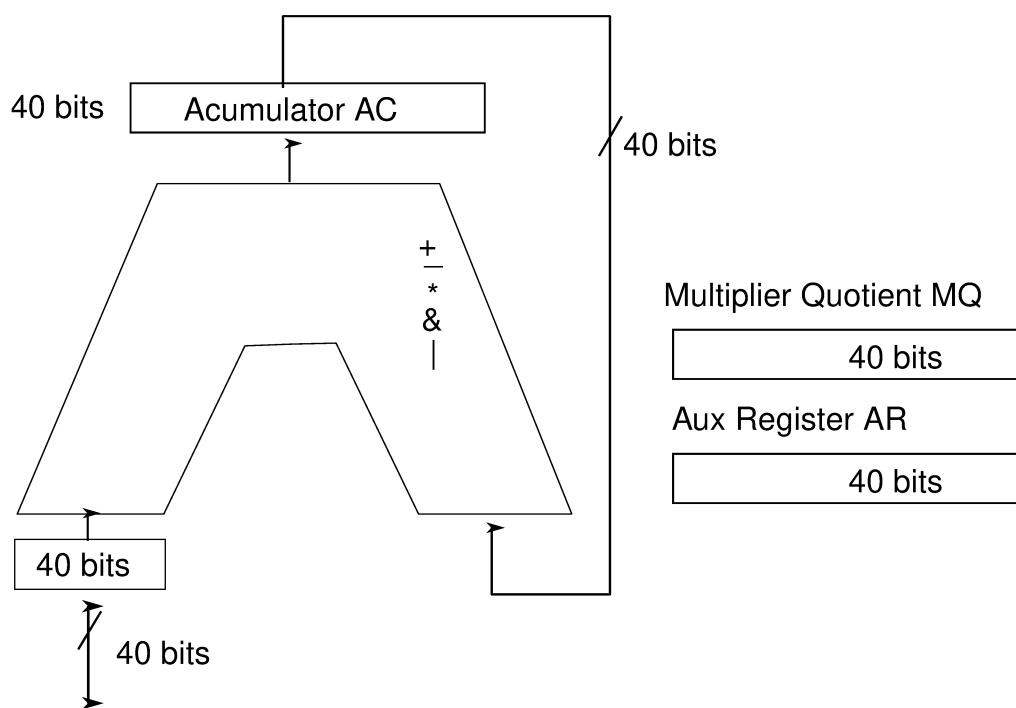
El programa se encuentra codificado en lenguaje binario

El programa es una secuencia de instrucciones y datos

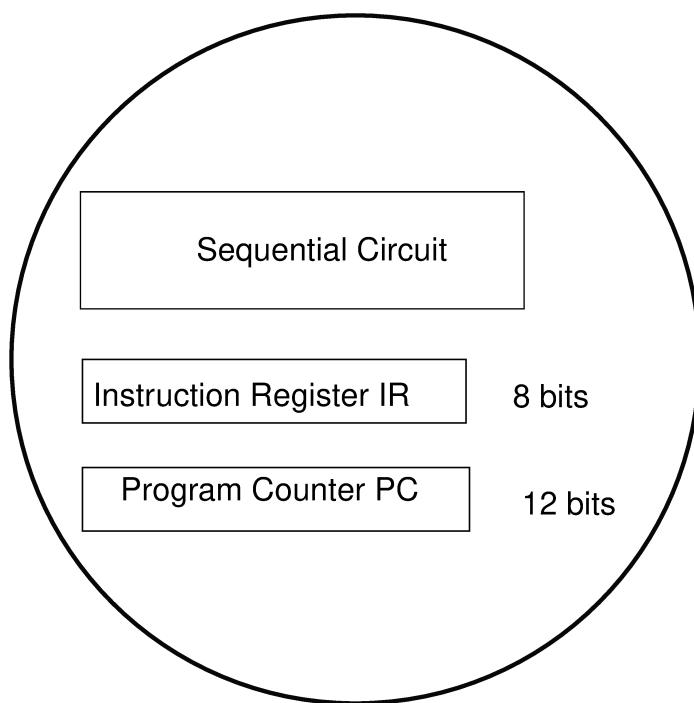
Los puertos de la memoria son MAR y MBR

La dirección de memoria en código binario es la entrada del circuito decodificador que activa una de sus salidas

ALU de la máquina IAS



Unidad de Control de la máquina IAS



La unidad de control es el circuito digital microelectrónico que lleva a efecto el ciclo de instrucción de cada instrucción ciclo de instrucción: cada instrucción pasa por 3 fases

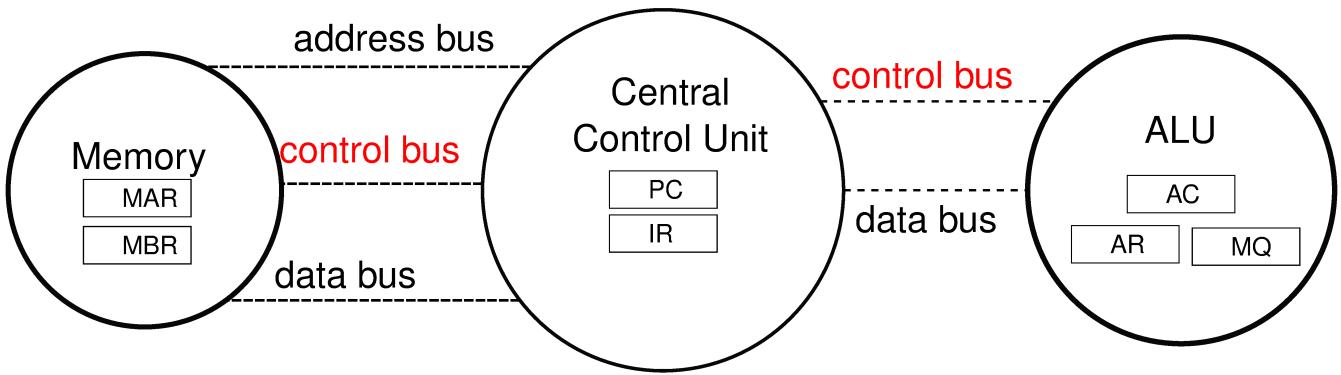
- captura de la instrucción por la unidad de control
- interpretación de la instrucción por la unidad de control
- ejecución de la instrucción por la unidad de control

circuito secuenciador: electrónica para la secuencia de fases del ciclo de instrucción

IR: almacena la instrucción a interpretar

PC: apunta a la siguiente instrucción que tiene que comenzar su ciclo de instrucción

Unidad de Control de la máquina IAS



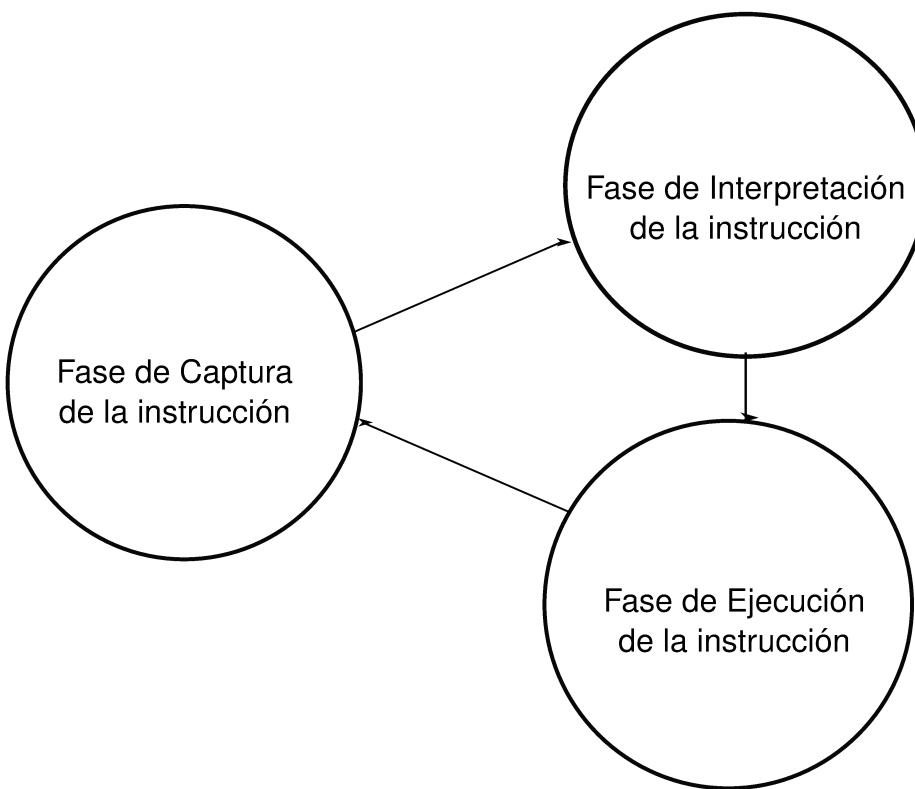
bus de datos: 40 hilos

bus de direcciones: 12 hilos

bus de control (memoria): micro-órdenes de lectura y escritura

bus de control (ALU): micro-órdenes de operaciones de suma, resta, and, or, etc..

Ciclo de instrucción



- Ejecución de un programa en la máquina IAS de von Neumann
 - Las instrucciones se ejecutan una detrás de otra, secuencialmente
 - Todas las instrucciones pasan por las distintas fases del ciclo de instrucción.

Ciclo de instrucción: Animación

- Programa en lenguaje máquina
- 3 instrucciones
- Ruta de datos: transferencia de instrucciones y de datos a través de los registros y los buses
- Buses externos al procesador: buses de conexión entre el chip procesador y la tarjeta de memoria → buses de direcciones, datos y lectura/escritura
- Buses internos al procesador: microbuses entre registros, ALU y unidad de control

- Se va a visualizar la transferencia de información entre registros a través de la ruta de datos de la computadora
- [Animación del ciclo instrucción](https://www.youtube.com/watch?v=04UGopESS6A) [https://www.youtube.com/watch?v=04UGopESS6A]

Tema 3 : Representación Digital de la Información: los DATOS

Índice

- Información: números, caracteres, imagen, sonido, etc ..
- Prefijos
- Digitalización de las señales
- Números
 - Sistemas posicionales: base 10 (decimales), base 2 (binaria), base 16 (hexadecimal)
 - Naturales: bases 10,2,8,16 . Conversión entre bases
 - Enteros: Signo Magnitud, Complemento a la base
 - Operaciones aritméticas: Suma,Resta
 - Operaciones lógicas: not, and, or, xor, desplazamiento de bits.
- Caracteres
 - Alfanuméricicos y Signos de Puntuación
 - ASCII standard y extendido

Representación Científica y Prefijos de las Unidades

Table 4. Prefijos

Prefijos	Tera	Giga	Mega	Kilo	mili	micro	nano	pico
Base 10 → magnitudes:m,gr,Hz, ..	10^{12}	10^9	10^6	10^3	10^{-3}	10^{-6}	10^{-9}	10^{-12}
Base 2 → magnitudes: Byte	2^{40}	2^{30}	2^{20}	2^{10}	—	—	—	—

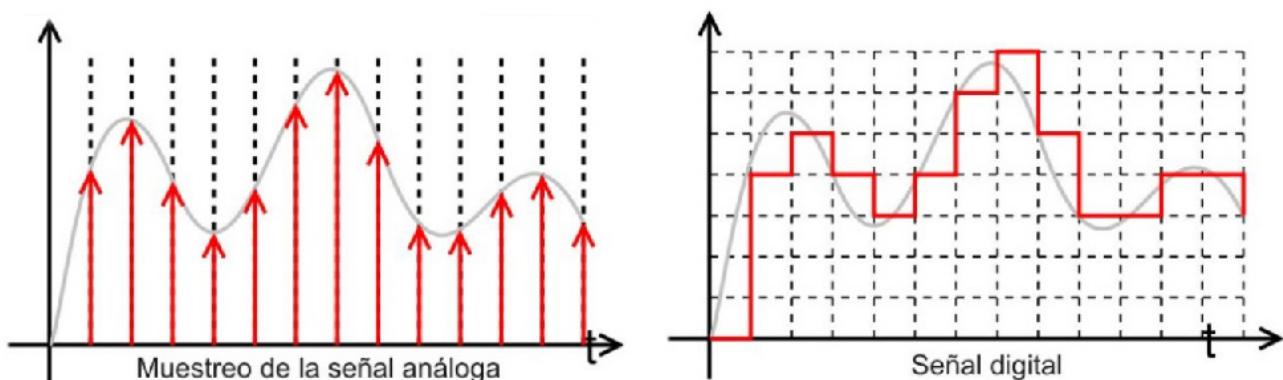
- Ejemplo: representar la magnitud=1000000000Hz debidamente
 - Notación científica → 10^9 Hz
 - Debidamente: Notación científica con prefijos f=1GHz → T=1/f= 10^{-9} seg= 1ns
- Ejemplo: capacidad de la memoria en Bytes
 - 1KB → 1 Kilo-Byte → 2^{10} Bytes → 1024 Bytes
 - 16KB → 16 Kilo-Bytes → 2^4 * 2^{10} Bytes → 2^{14} Bytes → 4096 Bytes
 - 1MB → 1 Mega-Byte → 2^{20} Bytes → 1048576 Bytes

Señales: Conversión Analógica Digital

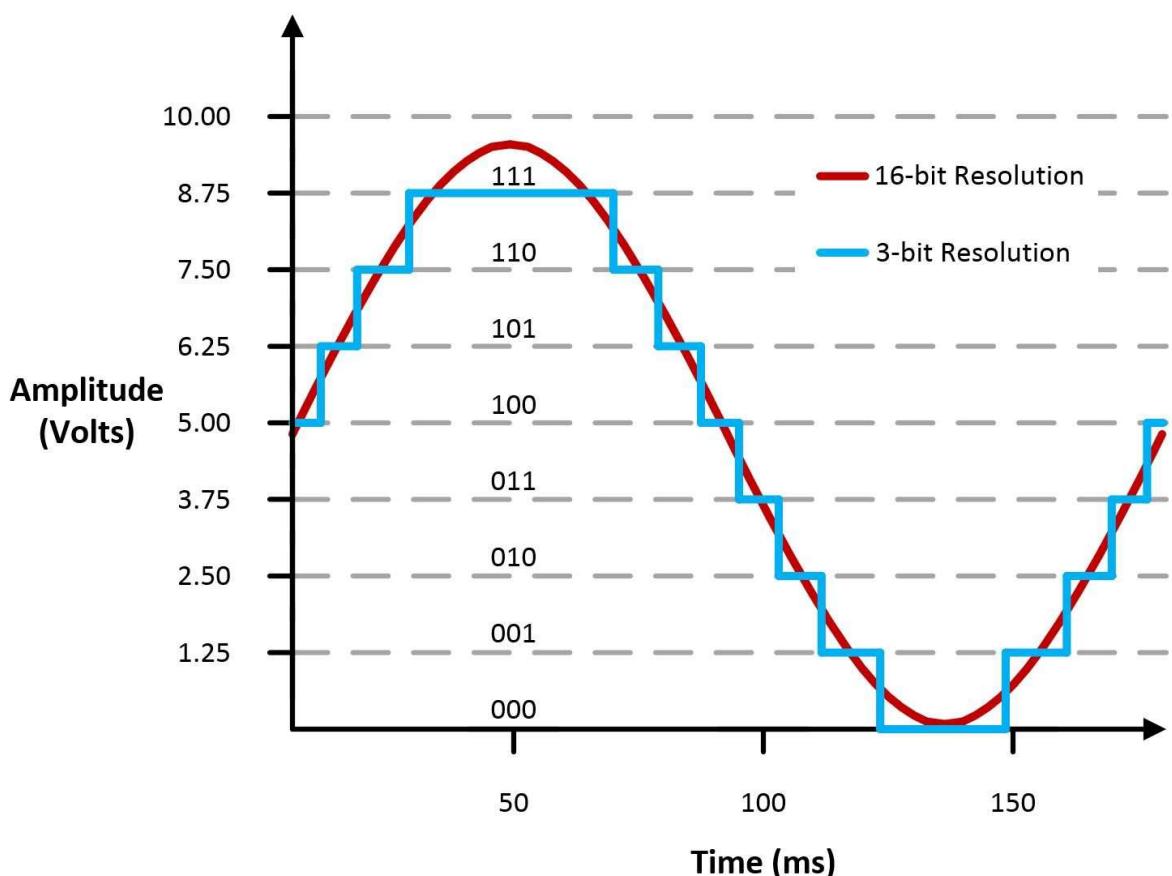
Analogica vs Digital

- Señal Continua
 - Amplitud: ∞ valores posibles en el rango
 - Tiempo: ∞ valores posibles en el rango
- Señal Discreta ó Digital
 - Amplitud: finitos valores posibles en el rango
 - Tiempo: finitos valores posibles en el rango

Señales : Muestreo y Cuantificación



Codificación

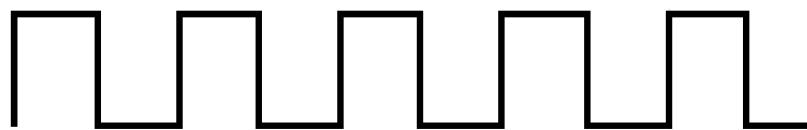


Calcular para las resoluciones de 3 bit y 16 bits cual es el mínimo incremento de señal codificable o error de

cuantificación: con 3 bits el número de niveles es $2^3=8$ niveles y el mínimo relativo es $2^{-3}=1/8$; con 16 bits el número de niveles es 2^{16} y el mínimo relativo es $2^{-16}=1/65536$.

Representación de los números en código binario : [Tema 3 : Representación Digital de la Información: los DATOS](#)

Señales Binarias : Abstractas

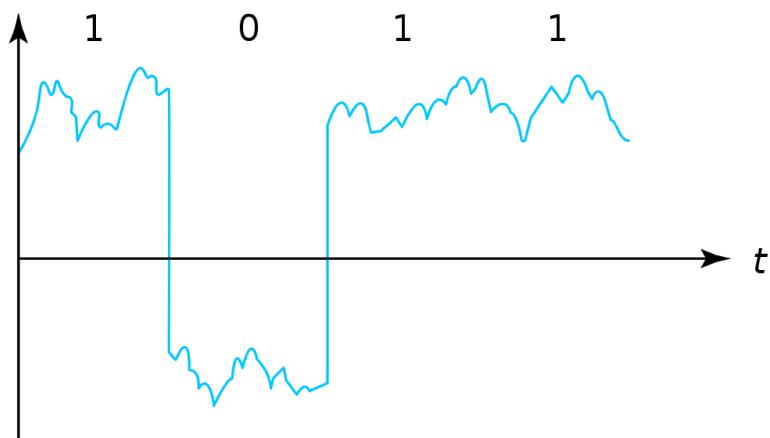


Eje ordenada: valores abstractos (0/1, High/Low, ON/OFF, etc ...).

Cronograma: Representación temporal de las señales digitales binarias.

Esa representación típica de los libros de texto, pizarra de clase, etc ... es ideal ya que físicamente siempre habrá distorsión.

Señales Binarias : Físicas



Eje ordenada: magnitudes físicas (mV ó mA).

La señal física está distorsionada por causas como pej: línea larga de transmisión (efectos capacitivos e inductivos).

Un ejemplo típico de distorsión son los tiempos de subida y bajada, que no son nulos sino del orden de unos nanosegundos.

La distancia considerable entre los dos niveles (binario) a la entrada del receptor hace fácil la discriminación entre el '0' y el '1'.

Digitalización de las Señales

Ventajas

- Calidad: Fácil de recuperar a pesar de la distorsión
- Almacenamiento: Fiabilidad, Diversidad Formatos
- Compatibilidad: Diversidad de Equipos (PC, móvil, coche, etc)
- Procesamiento: Sencillo, Flexible
- Coste: Barato (componentes)

Abstracción

- Niveles: el 0 y el 1
- Lógica binaria
 - Matemáticas: Algebra de Boole

Tema 3 :Representación de los Caracteres

Representación de los Caracteres

- Tipos de Caracteres:
 - Alfanuméricos: a,b,...z,0,1,...9,A,B...Z
 - Signos de Puntuación: !"#\$%&/()=
 - de Control: Salto de Línea (\n), Find de Fichero (EOF), Fin de String (\00, ...)
- Formatos
 - ASCII: standard y extendido
 - Unicode: UTF-8

ASCII Standard

2	3	4	5	6	7	30	40	50	60	70	80	90	100	110	120
0:	0	@	P	`	p	0:	(2	<	F	P	Z	d	n	x
1:	!	1	A	Q	a	q	1:)	3	=	G	Q	[e	y
2:	"	2	B	R	b	r	2:	*	4	>	H	R	\	f	p
3:	#	3	C	S	c	s	3:	!	+	5	?	I	S]	{
4:	\$	4	D	T	d	t	4:	"	,	6	@	J	T	^	
5:	%	5	E	U	e	u	5:	#	-	7	A	K	U	_	s
6:	&	6	F	V	f	v	6:	\$.	8	B	L	V	`	~
7:	'	7	G	W	g	w	7:	%	/	9	C	M	W	a	DEL
8:	(8	H	X	h	x	8:	&	0	:	D	N	X	b	l
9:)	9	I	Y	i	y	9:	'	1	;	E	O	Y	c	m
A:	*	:	J	Z	j	z									
B:	+	;	K	[k	{									
C:	,	<	L	\	l										

```
D: - = M ] m }
E: . > N ^ n ~
F: / ? 0 _ o DEL
```

American Standard Code for Information Interchange

Alfabeto anglosajón

7 bits $\rightarrow 2^7 = 128$ caracteres : 0x00 hasta 0x1F son 32 caracteres de control y el resto alfanuméricos

En hexadecimal rango [0x00-0x7F]

En decimal rango [0-127]

Upna : 0x55706E61

año 2023: 0x61—6F2032303233

ASCII Extendido

- Para poder representar caracteres de otras culturas Europeas es necesario expandir el standard con 1 bit más
- ASCII 8 bits $\rightarrow 2^8 = 256$ caracteres
- [Python Interpreter Shell](https://www.programiz.com/python-programming/online-compiler/) [<https://www.programiz.com/python-programming/online-compiler/>]

```
ord('A')
hex(ord('A'))
hex(ord('\n'))
chr(65)
chr(0x41)
[hex(ord(c)) for c in "Hola"]
[chr(c) for c in [0x48, 0x6f, 0x6c, 0x61, 0x20, 0x4d, 0x75, 0x6e, 0x64, 0x6f]]
[hex(ord(c)) for c in "ñ"]
[hex(ord(c)) for c in "\n \t"]
```

- La ñ tiene el código ASCII 0xF1

Tema 3: Representación de los Números NATURALES

Representación de los Números Decimales

- Decimal
 - 10 dígitos : 0,1,2,3,4,5,6,7,8,9
 - Pesos con base 10 : 10^n donde n es la posición del dígito dentro del número
- Ejemplo: número 5421

Table 5. Número 5421

Representación:	los símbolos 5421			
Posiciones:	3	2	1	0
Pesos:	$10^3 \rightarrow 1000$	$10^2 \rightarrow 100$	$10^1 \rightarrow 10$	$10^0 \rightarrow 1$
Dígitos:	5	4	2	1

Valores : ponderación	5*1000=cinco mil	4*100=cuatro cientos	5*10=cincuenta	1*1=uno
Valor:	5*1000+4*100+5*10+1= cinco mil cuatrocientos cincuenta y uno			

Representación de los Valores Naturales en Código Binario

- ¿Número? ¿Valor? ¿Código? ¿Representación?
 - 2 dígitos : 0,1
 - Pesos con base 2 : 2^n donde n es la posición del dígito dentro del número:-1024-512-256-128-64-32-16-8-4-2-1...
- Ejemplo: número 0b1011

Table 6. Número 0b1011

Representación:	los símbolos 1011			
Posiciones:	3	2	1	0
Pesos:	$2^3 \rightarrow 8$	$2^2 \rightarrow 4$	$2^1 \rightarrow 2$	$2^0 \rightarrow 1$
Dígitos:	1	0	1	1
Valores : ponderación	$1*8=ocho$	$0*4=cero$	$1*2=dos$	$1*1=uno$
Valor:	ocho+cero+dos+uno= once			

Representación de los Valores Naturales en Código Binario

- ¿Cómo se representa en binario el valor 123.125? b1111011.001
- ¿Cómo se calcula el valor del número binario b1111011.001?
- Parte Entera: divisiones sucesivas por la base 2
- Parte Fracción: multiplicaciones sucesivas por la base 2

Representación de los Valores Naturales en Código Octal

- Dígitos: 0,1,2,3,4,5,6,7
- Posiciones y Pesos
- ¿Cómo se representa en octal el valor 123.125? 0o173.1
- ¿Cómo se calcula el valor del número octal 0o173.1?
- Parte Entera: divisiones sucesivas por la base 8
- Parte Fracción: multiplicaciones sucesivas por la base 8

Representación de los Valores Naturales en Hexadecimal

- Dígitos: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F _ el valor de A es 10, B→11, C→12, D→13, E→14, F→15
- Posiciones y Pesos

- ¿Cómo se representa en hexadecimal el valor 123.125? 0x7B.2
- ¿Cómo se calcula el valor del número octal 0x7B.2?
- Parte Entera: divisiones sucesivas por la base 16
- Parte Fracción: multiplicaciones sucesivas por la base 16

Calculadora de Python

- [Python Interpreter Shell](https://www.programiz.com/python-programming/online-compiler/) [https://www.programiz.com/python-programming/online-compiler/]

```
bin(123)
oct(123)
hex(123)
int(0b1111011)
int(0o173)
int(0x7B)
```

Conversiones entre el sistema binario y sistemas con base potencia de 2

- Conversión Binaria-Hexadecimal
 - base $16=2^4$
 - grupos de 4 bits empezando por la dcha
 - b1111011 → 111 - 1011 → 0x7B
- Conversión Hexadecimal-Binaria
 - grupos de 4 bits
- Conversión Binaria-Octal
 - base $8=2^3$
 - grupos de 3 bits empezando por la dcha
 - b1111011 → 1 - 111 - 011 → 0o173
- Conversión Octal-Binaria
 - grupos de 3 bits

Tema 4 : Operaciones Aritméticas con los Números NATURALES

Suma binaria

- Suma $10011011 + 00011011 = 10110110$

Llevadas -->

1 1 1 1

$$\begin{array}{r}
 1 0 0 1 1 0 1 1 \quad \text{--sumando} \\
 + 0 0 0 1 1 0 1 1 \quad \text{--sumando}
 \end{array}$$

Valor suma 1 0 1 3 2 1 3 2 <- ¿cómo se representan los valores de cada posición?

Resultado --> 1 0 1 1 0 1 1 0 <--suma

- **LLEVADA**

- Valor suma: el valor 3 de la posición 1 $\rightarrow 3 \cdot 2^1 = 6 \rightarrow 110$
 - el dígito 1 de la posición 1 es el resultado
 - el dígito 1 de la posición 2 es la LLEVADA
- Conclusión: Cuando la suma en una posición específica tiene un valor es mayor o igual a la base hay que restar **n** veces la base y el valor **n** será la llevada a sumar en la posición siguiente.

Resta binaria

- Resta $10110110 - 10011011 = 00011011$

Sumar crédito al minuendo

2 2 2 2

1 0 1 1 0 1 1 0 <--minuendo
- 1 0 0 1 1 0 1 1 <--sustraendo

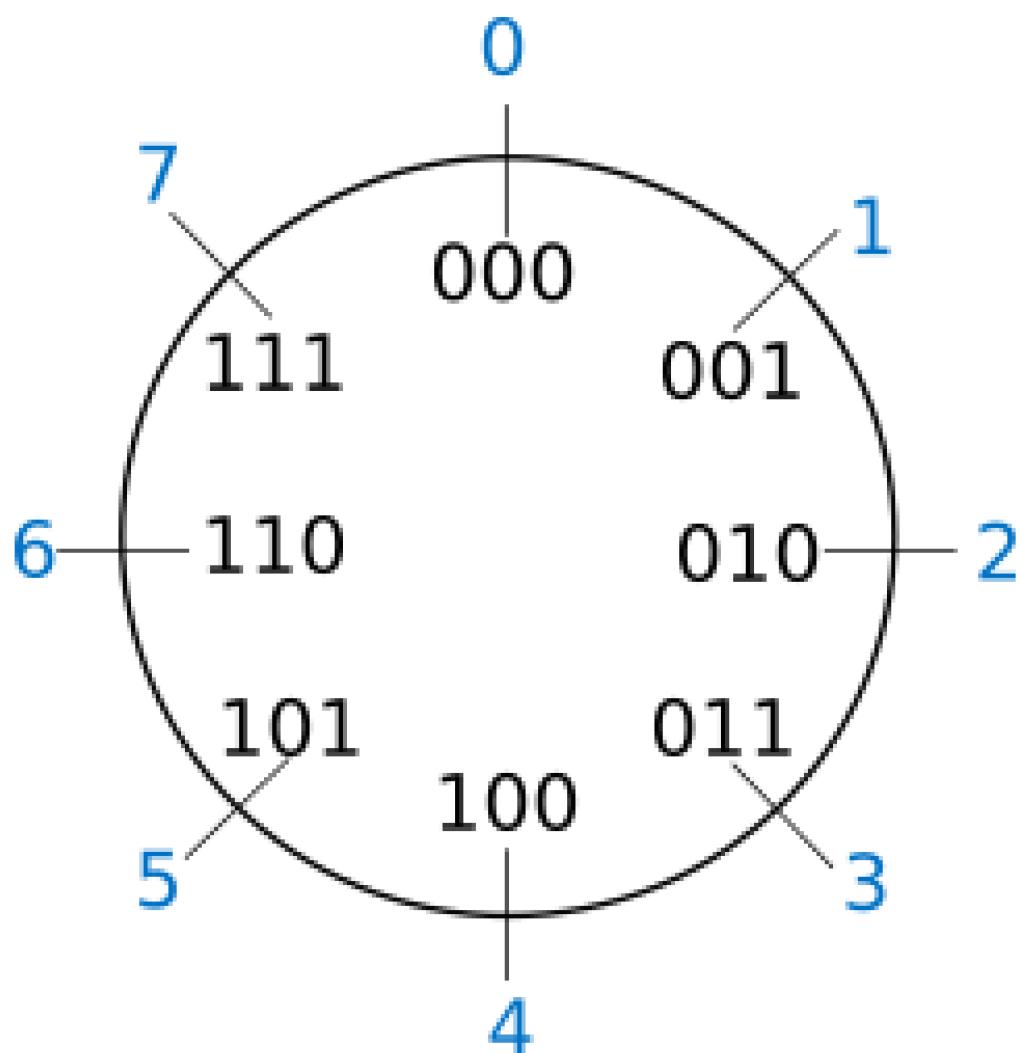
Sumar llevada al sustraendo

Resta

0 0 0 1 1 0 1 1

- Cuando en una posición específica el minuendo es menor que el sustraendo se suma la base al minuendo antes de realizar la resta y se suma la llevada al sustraendo de la posición siguiente.

Aritmética Modular: la rueda



Representación binaria de números con 3 dígitos.
 2^3 : 8 combinaciones posibles, → MODULO 8
 Ejemplo:
 cuentakilómetros del coche.
 Ejemplo: registro de 3 celdas → limitado a 8 combinaciones posibles.
 ¿Cuál es la siguiente combinación a 111?,
 $111+1 = ?$.
 Calcular la representación del valor 33 en módulo 8 → Resto($33/8$)=1 → en binario 001
 33 pasos en la rueda equivale al número 001 → aritmética modular en módulo 8

Operaciones aritméticas: Hexadecimal

- Base hexadecimal
 - $0xD1B + 0xFF = 0x181A$
 - $0xE53 - 0xBA9 = 0x2A9$

Tema 3 :Representación de los Números ENTEROS

Representación de Números con Valores Enteros

- Signo-Magnitud
- Complemento a la base

Representación en Signo-Magnitud

- Signo → un bit (Binary digit)
- Base 2 :
 - valores positivos: el signo el **bit 0** en la posición MSB (More Significant Bit) y resto de bits representa el mód

- valores negativos: el signo el **bit 1** (base-1) en la posición MSB (More Significant Bit) y resto de bits representa el módulo
- Ejemplo +123 → 0b01111011 y -123 → 0b11111011
- Dibujar la tabla y la rueda con todos los valores con sus representaciones.
- ¿Cuantas representaciones son posibles? ¿Es simétrico el rango de valores representado? ¿Cuantas representaciones tiene el cero?
- ¿Cuál es la relación entre los números positivos y negativos?
- Extender el número de bits del número sin cambiar su valor

Extensión del signo en Signo-Magnitud

- Números positivos
 - ¿Cuánto vale un cero a la izda?
 - Extender el valor '+123' a 12 bits → 000001111011
- Números negativos
 - ¿Cuánto vale a la izda?
 - Extender el valor -123 a 12 bits → Convertimos el valor positivo en negativo
 - 100001111011

Representación en complemento a la base 2 : C2

- Signo → un dígito
- Base 2: Complemento a 2 → **C2**
- Valores positivos: Definición: igual que los valores positivos en código Signo-Magnitud
- Valores negativos: Definición: Hay que restar el código del valor en positivo del minuendo 0000000 (base)
 - Ejemplo '+123' → 0b01111011 y -123 → **00000000-01111011** = 0b100000101
 - **Alternativa 1** a la definición: El código del valor negativo se puede calcular invirtiendo los bits del código del valor positivo y después sumarle 1
 - Equivale a calcular el C1 y sumarle 1
 - **Alternativa 2** a la definición: El código del valor negativo se puede calcular a partir del código del valor positivo
 - empezando por la dcha repetir los bits hasta el primer uno e invertir el resto de bits

Representación en complemento a la base 2 : C2

- ¿Cuál es la relación entre los números positivos y negativos?
 - El C2 de un número positivo es el código C2 de su valor en negativo
 - El C2 de un número negativo es el código C2 de su valor en positivo
 - Dibujar la tabla y la rueda con todos los valores con sus representaciones.
 - ¿Cuantas representaciones son posibles? ¿Es simétrico el rango de valores representado? ¿Cuantas representaciones tiene el cero?
 - Extender el número de bits del número sin cambiar su valor → Extensión del bit de SIGNO

Extensión del signo en C2

Table 7. Razonamiento de la extensión de signo de un número negativo: números de 3 bits

Valor	C2 sin extensión	C2 con extensión
+33	0100001	00100001
-33	0000000 -0100001 ----- 1011111	00000000 -00100001 ----- 11011111

- Aplicamos la definición para los dos ejemplos, el segundo caso es una extensión del primero ya que hemos añadido un dígito más al minuendo y al sustraendo:
 - Se observa que en el C2 con extensión, al hacer la resta y extender con un 0 más el minuendo y el sustraendo, provoca la extensión con un bit más en la resta de valor 1 en el dígito más significante. Según añado ceros al minuendo y sustraendo, aparecen unos en la resta sin alterar su valor.

Asimetría del rango en C2: -4 con 3 bits

- Con números de 3 bits el formato S-M es simétrico con valores en el rango (+3,-3), en cambio el formato C2 tiene el rango (+3,-4)
- En C2 el valor +4 se representa como 0b0100 y necesita por lo tanto 4 bits, no se puede representar con 3 bits, y el valor -4 se representa con el C2(0100), es decir, 1100 también con 4 bits. El 1100 se puede comprimir ya que tiene el signo extendido con la repetición de 1 de bit más significativo, por lo que la representación 100 es la representación del -4

Complemento a 2 : Ejemplos

- 0b101010101 está en C2 → ¿Cuál es su valor?
 - como es negativo no es un sistema posicional
 - tenemos que calcular el valor negativo a través del valor positivo
 - La representación del valor positivo es el C2 del valor negativo
 - $C2(0b101010101) = 0b010101011$ cuyo valor es $2^7+2^5+2^3+2^1+2^0=128+32+8+2+1=+171$
 - El valor de 0b101010101 es -171
- Si la representación de -123 es 0b100000101 ¿cuál es la de '+123' ?
 - $C2(0b100000101)=0b01111011$ representa el valor '+123'

Número en complemento a 2 y base hexadecimal



Un número binario se puede representar en hexadecimal y hacer la interpretación en complemento a 2. Hay que tener cuidado con las extensiones del signo

- Calcular el valor del número 0xAAA si dicho número tiene formato en complemento a 2
 - si lo convertimos a binario el número empieza por 1, luego es negativo
 - para saber su valor calculo su complementario C2 y tendrá la representación del positivo
 - $0x000-0xAAA = 0x556 \rightarrow 5*16^2+5*16^1+5*16^0 = 5*256+5*16+5 = 1280+80+5 = '+213' \rightarrow 0xAAA$ tiene de valor -213

Número en complemento a 2 y base hexadecimal

- Realizar la suma de los números en formato complemento a 2: 0x80+0x80
 - sumar sin extender el signo de los operandos ¿Hay overflow?



Extender el número 0x80. ¿ Por qué hay que tener cuidado ?

- sumar extendiendo un dígito el signo de los operandos 0x80

Extensión del signo en C2: problema de la BASE

Table 8. Extensión del Signo del Nº 0x80 en C2 en binario, hexadecimal y octal

NºBits	Binario	Hexadecimal
8	10000000	1000_0000 → 0x80
9	110000000	1111_1000_0000 → 0xF80
10	1110000000	1111_1000_0000 → 0xF80
11	11110000000	1111_1000_0000 → 0xF80
12	111110000000	1111_1000_0000 → 0xF80
13	1111110000000	1111_1111_1000_0000 → 0xFF80

Tema 4 :Operaciones Aritméticas con los números ENTEROS

Operaciones Aritméticas: Suma y Resta

Operaciones aritméticas en C2

- Suma
 - Se realiza de la misma manera como se ha visto para números naturales.
 - Si hay llevada en el MSBit, NO se tiene en cuenta, se elimina.
 - A=0b11011011. Suma A+A

Llevadas -> 1 1 1 1 1 1

$$\begin{array}{r} 11011011 \\ + 11011011 \\ \hline \end{array} \quad \text{(Valor -37)}$$

Valor suma 2 1 3 2 1 3 2

Resultado --> 1 0 1 1 0 1 1 0 --(Valor -74)

- Resta
 - La resta de números con signo se puede realizar de dos formas: A-B ó A-B = A+(-B)

- A = 0b00110110 y B = 0b10011011
- Si hay llevada en el MSBit, no se tiene en cuenta, se elimina.

```

Crédito 2 2 2 2
              1 0 1 1 0 1 1 0<--(Valor -74)
              - 1 1 0 1 1 0 1 1<--(Valor -37)

LLevada 1 1 1 1 1 1
*****  

Resta    1 1 0 1 1 0 1 1 (Valor -101)

```

Operaciones aritméticas C2: Overflow o Desbordamiento

- A = 0b00110110 y B = 0b10011011 → Calcular A-B
- Con 8 bits el máximo valor es 01111111 de valor $2^7-1=128-1=127$
- La resta A-(B)=A+(-B)=54+103=157>127 → **Overflow o Desbordamiento**

```

Crédito 2 2 2 2
              0 0 1 1 0 1 1 0<--(Valor = 54)
              - 1 0 0 1 1 0 1 1<--(Valor = -103)

LLevada 1 1 1 1
*****  

Resta 1 0 0 1 1 0 1 1 (Valor -101)

```

- El valor -101 en lugar de la resta correcta +157 es debido a que el resultado esta fuera de rango →
- Observarmos que hemos hecho la SUMA de dos números POSITIVOS y el resultado ha sido NEGATIVO

Operaciones aritméticas C2: Overflow



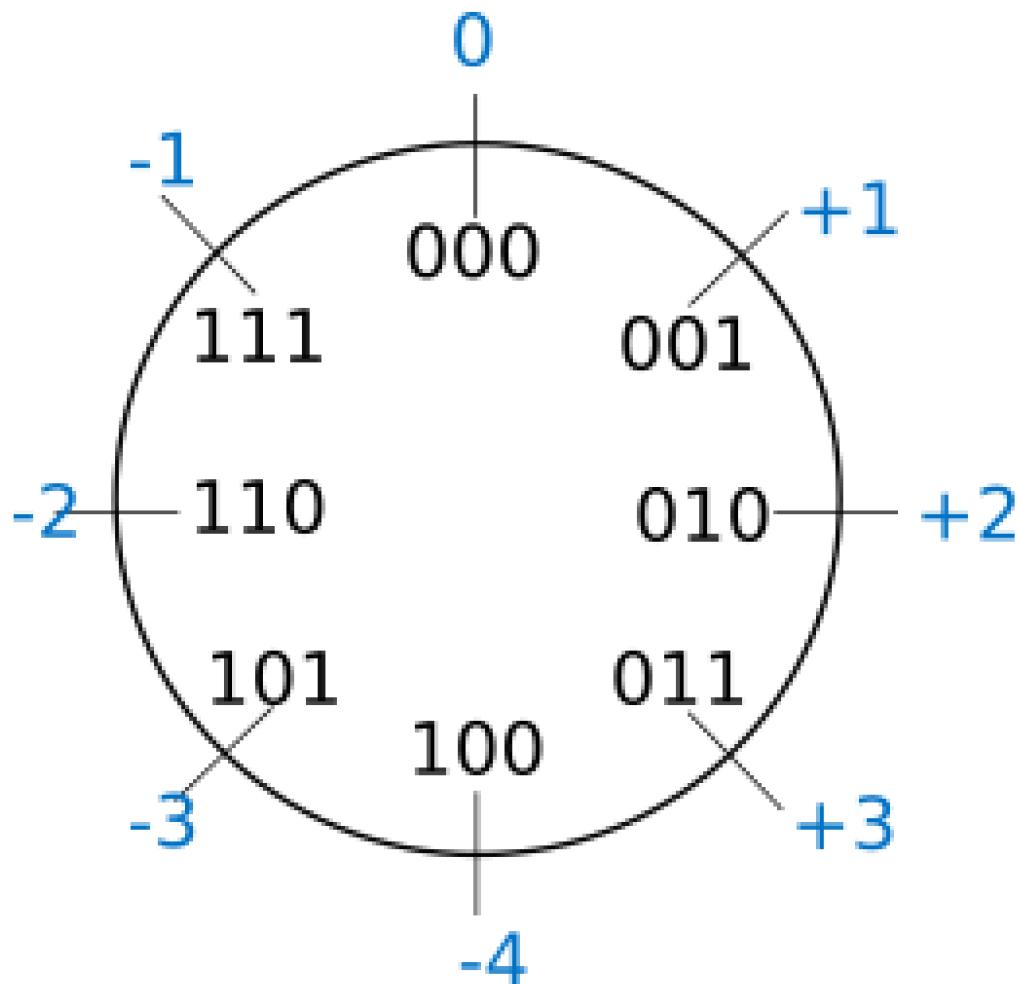
Al realizar la suma de dos valores con el mismo signo si el resultado es de signo contrario hay overflow

Operaciones aritméticas C2: Overflow

- Overflow: la operación requiere operandos con mayor número de bits manteniendo el valor para que el resultado sea correcto.
- Si dos operandos a sumar tienen diferente signo nunca hay overflow
- Si dos operandos a sumar tienen el mismo signo y resultado tiene signo contrario : **Error de Overflow**.
- Ejemplo:
 - Operandos de 1 byte : 01111111+01111111=11111110 → sumandos positivos y resultado negativo

- Solución: **Extensión del signo** : Operandos 9 bits $\rightarrow 001111111 + 001111111 = 011111110$
- la repetición del bit más significativo no altera el valor de la representación
- el bit más significativo es 0 si es positivo y 1 si es negativo. Por lo tanto, 01010 equivale a 01010 ó 001010 ó 0....0001010. Por lo tanto, 1010 equivale a 11010 ó 111010 ó 1....111010

C2: Representación gráfica del Overflow

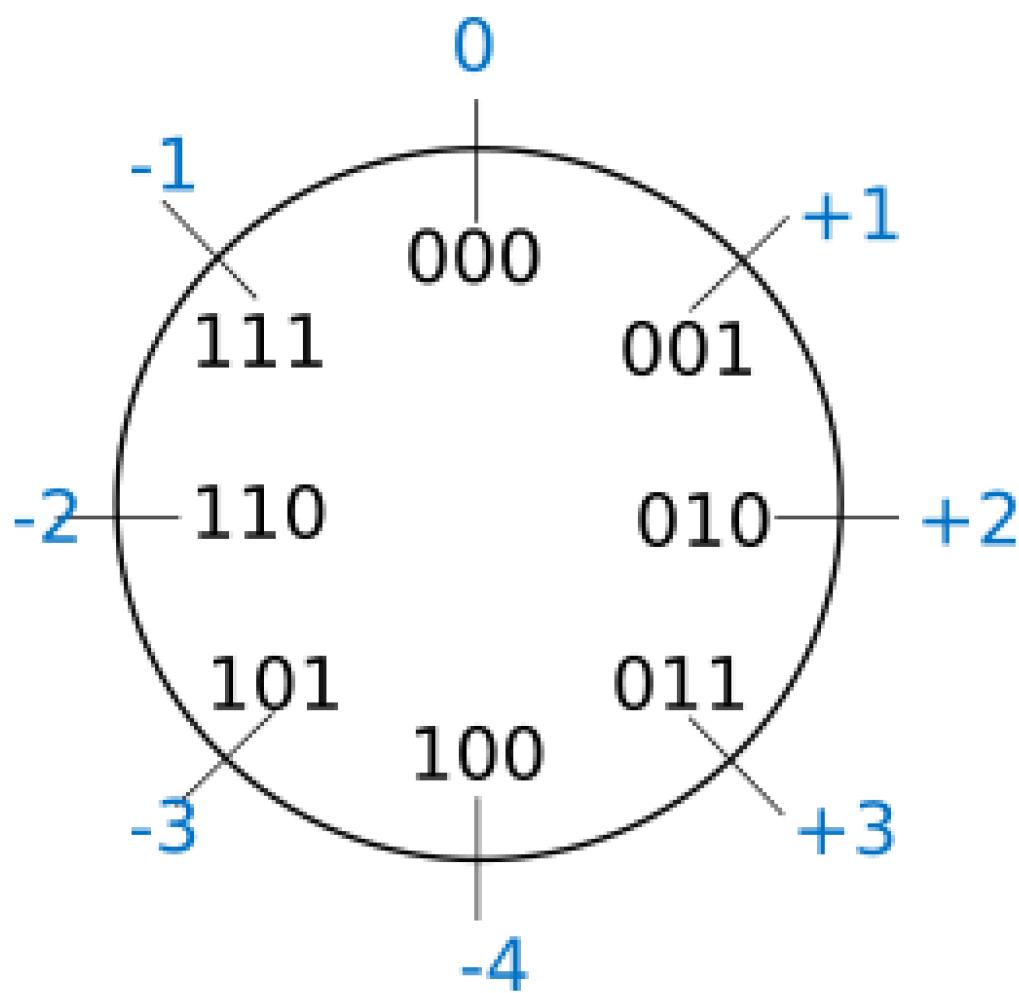


Si a partir de la posición 010 nos movemos dos posiciones en sentido horario llegamos a la posición 100.

Si a 010 le sumamos el valor 2 nos da como resultado 100

Por lo tanto
 $010 + 010 = 100$, es decir,
 $2 + 2 = -4 \rightarrow \text{overflow}$ ya que el +4 necesita 4 bits y estamos trabajando con 3 bits únicamente.

Aritmética Modular de valores representados en Complemento a 2



Representación de números binarios de 3 bits en C2
 Operaciones de suma y resta modular → método gráfico
 A partir de la posición 001 si nos movemos en sentido horario (SUMA modular) 2 posiciones obtenemos la posición 011, es decir, $1+2=3$
 A partir de la posición 110 si nos movemos en sentido horario (SUMA modular) 9 posiciones obtenemos la posición 111, es decir, $-2+9=-1$
 A partir de la posición 110 si nos movemos en sentido antihorario (RESTA modular) 4 posiciones obtenemos la posición 010, es decir, $-2-4=+2$
 Los errores de **overflow** se resuelven aumentando el número de bits de la

representación, pero siempre existirá un rango que si lo traspasamos dará overflow.

Comparación S-M, y C2

Table 9. Números de 3 bits

Valor	S-M	C2
+3	011	011
+2	010	010
+1	001	001
0	000	000
	100	---
-1	101	111
-2	110	110
-3	111	101
-4	-	100

Tema 4 :Operaciones Lógicas

- Operadores
 - Operadores aritméticos: suma, resta, multiplicación, ...

- Operadores lógicos: or (suma), and (producto), negación, or exclusiva, etc...
- Operadores comparadores: > (...mayor que..), < (...menor que..), == (...igual a..), etc
- Operadores de desplazamiento: >> (desplazar hacia la derecha), << (desplazar hacia la izquierda)

Tablas de la Verdad de los operadores NOT, OR, AND, XOR

Table 10. NOT

x	$z = \bar{x}$
0	1
1	0

Table 11. OR

x	y	$z = x + y$
0	0	0
0	1	1
1	0	1
1	1	1

Table 12. AND

x	y	$z = x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

Table 13. XOR

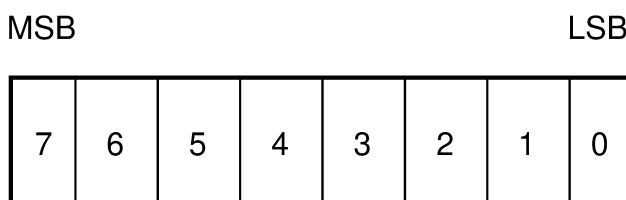
x	y	$z = x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Tema 5: Representación Digital de la Información: las INSTRUCCIONES

Registro

- Un registro es un circuito que funciona como una unidad de memoria y que almacena un único dato o una instrucción máquina.
- Los registros:

- 'almacenan' una palabra formada por una secuencia de bits.
- son una array de celdas en una dimensión, donde cada celda almacena un bit.
- Su tamaño normalmente es un múltiplo de 8 bytes y recibe un nombre para poder ser referencia, por ej RAX
 - 8 bits: 1 Byte
 - 16 bits: Word. Por razones históricas.(recordad que el tamaño de una palabra en otro contexto depende de la máquina de que se trate)
 - .
 - 32 bits: double word
 - 64 bits: quad word
- Las celdas se enumeran empezando por cero.
- LSB: Least Significant Bit es el bit de menor peso
- MSB: Most Significant Bit es el bit de mayor peso



Lenguaje RTL: operador transferencia

- El lenguaje RTL es un Lenguaje para la descripción de INSTRUCCIONES MAQUINA: Register Transfer Language (RTL)
- El lenguaje RTL tiene como objetivo poder expresar las instrucciones máquina que ejecuta la CPU como sumar(ADD),restar(SUB),mover(MOV), etc. La descripción se realiza a nivel de transferencia de datos entre 'registros' internos de la CPU o entre registros internos y la memoria externa.
- La operación de **transferencia** se representa con una flecha de derecha a izquierda
 - Operador transferencia \leftarrow
 - Sentencia transferencia: $R2 \leftarrow R1$
 - A $R1$ se le llama registro fuente y a $R2$ registro destino
 - Interpretación: Copiamos o Transferimos el contenido del registro $R1$ en el registro $R2$

Lenguaje RTL: otras sentencias

Sentencia Condicional:

If ($K1=1$) then $R2 \leftarrow R1$

$K1:R2 \leftarrow R1$

La transferencia o copia se realiza únicamente si $K1$ es verdad, es decir, si $K1$ vale el valor lógico 1 (TRUE).

Sentencia Concurrente:

Operador coma

$K3:R2 \leftarrow R1, R3 \leftarrow R1$

Si $K3$ es verdad el contenido de $R1$ se copia en $R2$ y $R3$

Sintaxis de las instrucciones en el lenguaje INTEL

- El formato de las instrucciones en lenguaje ensamblador se conoce como 'sintaxis' de las instrucciones.
- SINTAXIS ASM: Etiqueta-Código de Operación- Operando1- Operando2- Comentario
- Arquitecturas x86-64 y x86

Table 14. Sintaxis Intel: Estructura

label:	op_mnemonic	operand_destination	,	operand_source	#comment
--------	-------------	---------------------	---	----------------	----------

Sintaxis de las instrucciones en el lenguaje INTEL : Ejemplo

- Ejemplo:

```
bucle: sub    rsp,16          ;RSP <- RSP-16. Resta
       je     bucle           ;je: jump equal:
                           ;salto si la última operación dió resultado
cero
suma:  add    eax,esi         ;EAX <- EAX+M[ESI] . Sumar
       mov    ax,[resultado]  ;AX <- M[resultado].
                           ;Copiar en el registro AX el contenido de la
                           ;posición de memoria resultado

resultado: "reserva de memoria"
```

Sintaxis de las instrucciones en el lenguaje de la compañía telefónica AT&T

- SINTAXIS ASM: Etiqueta-Código de Operación- Operando1- Operando2- Comentario
- Arquitecturas x86-64 y x86

Table 15. Sintaxis AT&T: Estructura

label:	op_mnemonic	operand_source	,	operand_destination	#comment
--------	-------------	----------------	---	---------------------	----------

- La gran diferencia con el lenguaje de Intel son el orden de los dos operandos
- Otra pequeña diferencia son los prefijos de los operandos para indicar el modo de direccionar el operando

Sintaxis de las instrucciones en el lenguaje AT&T : Ejemplo

- Ejemplo:

```
bucle: sub    $16,%rsp        ;RSP <- RSP-16. Resta
       je     bucle           ;je: jump equal:
                           ;salto si la última operación dió resultado
```

```

cero
suma: add    %esi,%eax          ;EAX <- EAX+M[ESI] . Sumar
      mov    resultado,%ax       ;AX <- M[resultado].
                                ;Copiar en el registro AX el contenido de la
                                posición de memoria resultado

resultado: "reserva de memoria"

```

Sintaxis AT&T

La sintaxis del lenguaje ensamblador depende del "traductor" del proceso de ensamblaje (**assembler**) utilizado, en este caso, se utiliza el assembler GAS.

ETIQUETA: Se especifica en la primera columna. Tiene el sufijo ":"

CODIGO DE OPERACION: Se utilizan símbolos mnemónicos que ayudan a interpretar intuitivamente la operación.

Pej: ADD sumar, MOV mover, SUB restar, ...

OPERANDO FUENTE Y/O DESTINO:

dato alfanumérico: representación alfanumérica → 16

direcciónamiento inmediato: prefijo \$

dirección de memoria externa: etiqueta → resultado

direcciónamiento directo

registros internos de la CPU: %rax,%rbx,%rsp,%esi,..

El prefijo "%" significa que el nombre hace referencia a un registro

tamaño del dato operando: sufijos de los mnemónicos:

q(quad):8 bytes, l(long):4 bytes, w(word):2 bytes, b(byte):1 byte.

Sin sufijo se toma la limitación del tamaño del registro referenciado
y si no hay limitación el traductor avisa del error

Operandos: Modo de Direcciónamiento

- Direcciónamientos:

INMEDIATO: El valor del operando está ubicado inmediatamente después del código de operación de la instrucción. Únicamente se especifica el operando fuente.

sintaxis: el valor del operando se indica con el prefijo \$.

ejemplo: **movl \$0xabcd1234, %ebx.** El operando fuente es el valor 0xABCD1234

REGISTRO: El valor del operando está localizado en un registro de la CPU.

sintaxis: Nombre del registro con el prefijo %.

ejemplo: **movl %eax, %ebx.** El operando fuente es el REGISTRO EAX y el destino es el REGISTRO EBX

DIRECTO: La dirección efectiva apuntando al operando almacenado en la Memoria Principal es la dirección absoluta referenciada por la etiqueta especificada en el campo de operando. El programador utiliza el direcciónamiento directo pero el compilador lo transforma en un direcciónamiento relativo al contador de programa. Ver direcciónamiento con desplazamiento.

sintaxis: una etiqueta definida por el programador

ejemplo: **je somePlace .** Salto a la dirección marcada por la etiqueta somePlace si el resultado de la operación anterior activa el flag ZF=1 del registro RFLAG.

INDEXADO: El valor del operando está localizado en memoria. La dirección efectiva apuntando a Memoria es la SUMA del valor del registro_base MAS scale POR el valor en el registro_índice, MAS el offset. 'EA=Offset+R_Base+R_índice*Scale'

sintaxis: lista de valores separados por coma y entre paréntesis (base_register, index_register, scale) y precedido por un offset.

ejemplo: **movl \$0x6789cdef, -16(%edx, %eax, 4)** . La dirección efectiva del destino es **EDX + EAX*4 - 16**.

Operandos: Modo de Direccionamiento

- Direccionamientos:

INDIRECTO: Si el modo general de indexación lo particularizamos en (base_register) entonces la dirección del operando no se obtiene mediante una indexación sino que la dirección efectiva es el contenido de rdx y por lo tanto se accede al operando indirectamente.

sintaxis: (base_register)

ejemplo: **movl \$0x6789cdef, (%edx)** . La dirección efectiva del destino es EDX. EDX es un puntero.

RELATIVO: registro base más un offset: El valor del operando está ubicado en memoria. La dirección efectiva del operando es la suma del valor contenido en un registro base más un valor de offset.

sintaxis: registro entre paréntesis y el offset inmediatamente antes del paréntesis.

ejemplo: **movl \$0xaabbccdd, -12(%eax)** . La dirección efectiva del operando destino es EAX-12

Operandos: Modo de Direccionamiento

- Direccionamientos **prohibidos**
 - En la misma instrucción los campos operando fuente y el operando destino no pueden hacer referencia a direcciones de memoria
- Ejemplos de direccionamientos prohibidos ó erróneos:
 - mov dato,suma → siendo dato y suma dos direcciones de la memoria
 - mov (%edx),suma → siendo EDX un puntero a memoria y suma una dirección de memoria

Modos de direccionar los operandos: Ejemplos

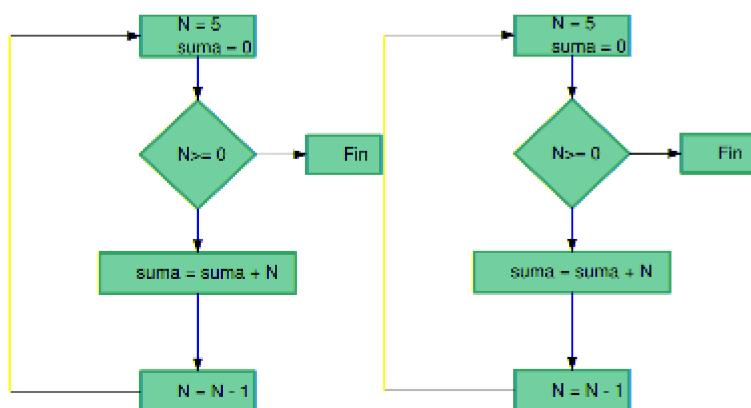
Table 16. Modos de Direccionamiento de los Operandos

Direccionamiento Operando	Valor Operando	Nombre del Modo
\$0	Valor Cero	Inmediato
%rax	RAX	Registro
loop_exit	M[loop_exit]	Directo
data_items(,%rdi,4)	M[data_item + 4*RDI]	Indexado
(%rbx)	M[RBX]	Indirecto
(%rbx,%rdi,4)	M[RBX + 4*RDI]	Indirecto Indexado

- M[loop_exit]: directo ya que loop_exit es una dirección de memoria externa y M indica la memoria externa.
- M[RBX]: indirecto ya que RBX es una dirección de memoria interna y M indica memoria externa: A la mem. externa se accede a través de la mem. interna.

Programa sum1toN: Organigrama

- Calcular la suma de los 5 primeros números naturales



Programa sum1toN en lenguaje C

```

/*
Programa:      sum1toN.c
Descripción:   realiza la suma de la serie 1,2,3,...N

Arquitectura del Procesador: La programación de este algoritmo en lenguaje C NO
DEPENDE de la arquitectura del procesador ¿Por qué y Cómo es posible?
Lenguaje:      C99
Descripción:   Suma de los primeros 5 números naturales
Entrada:       Definida en una variable
Salida:        Sin salida
Compilación:   gcc -m32 -g -o sum1toN sum1toN.c -> -g: módulo binario depurable
                -> -m32: módulo binario
arquitectura x86-32 bits
S.O:           GNU/linux 4.10 ubuntu 17.04 x86-64
Librería:      /usr/lib/x86_64-linux-gnu/libc.so
CPU:           Intel(R) Core(TM) i5-6300U CPU @ 3.0GHz
Compilador:    gcc version 6.3
Ensamblador:  GNU assembler version 2.28
Linker/Loader: GNU ld (GNU Binutils for Ubuntu) 2.28
Asignatura:    Estructura de Computadores
Fecha:         20/09/2017
Autor:         Cándido Aramburu
*/

```

```

#include <stdio.h> // cabecera de la librería de la función printf()

// función de entrada al programa
void main (void)
{
    // Declaración de variables locales
    char suma=0;
    char n=0b101;
    // bucle
    while(n>0){
        suma+=n;
        n--;
    }
    printf("\n La suma es = %d \n",suma);
}

```

Programa sum1toN para la máquina x86 en lenguaje AT&T

- Lenguaje ensamblador ATT para la arquitectura x86-32

```

### Programa: sum1toN.s
### Descripción: realiza la suma de la serie 1,2,3,...N
### Arquitectura del Procesador: x86 32 bits
### Compilación
###     gcc -m32 -g -nostartfiles -o sum1toN sum1toN.s
### o como alternativa
###     Ensamblaje as --32 --gstabs sum1toN.s -o sum1toN.o
###     linker -> ld -melf_i386 -I/lib/i386-linux-gnu/ld-linux.so.2 -o sum1toN
sum1toN.o -lc

## Declaración de variables
.section .data

n: .int 5

.global _start

## Comienzo del código
.section .text
_start:
    mov $0,%ecx # ECX implementa la variable suma
    mov n,%edx
bucle:
    add %edx,%ecx
    sub $1,%edx
    jnz bucle

```

```
    mov %ecx, %ebx # el argumento de salida al S.O. a través de EBX según
convenio

    ## salida
    mov $1, %eax # código de la llamada al sistema operativo: subrutina exit
    int $0x80      # llamada al sistema operativo

.end
```

Programa sum1toN para la máquina x86 en lenguaje Intel

- Lenguaje ensamblador INTEL y assembler nasm

```
;;; Programa: sum1toN.asm
;;; Descripción: realiza la suma de la serie 1,2,3,...N
;;; Arquitectura del Procesador: x86 32 bits
;;; Lenguaje INTEL
;;; Assembler NASM

;;; nasm -hf -> ayuda de la opción f
;;; Ensamblaje nasm -g -f elf sum1toN.asm -o sum1toN.o
;;; linker -> ld -m elf_i386 -o sum1toN sum1toN.o

BITS 32 ; cpu MODE
; Declaración de variables
section .data

n: dd 5 ; 4 bytes

global _start

; Comienzo del código
section .text
_start:
    mov ecx,0 ; ECX implementa la variable suma
    mov edx,[n] ; EDX implementa es un alias de la variable n
bucle:
    add ecx,edx
    sub edx,1
    jnz bucle

    mov ebx, ecx ; el argumento de salida al S.O. a través de EBX según
convenio

    ; salida
    mov eax,1 ; código de la llamada al sistema operativo: subrutina exit
    int 0x80      ; llamada al sistema operativo
```

Tema 6: Programación en Lenguaje Ensamblador (x86): Construcciones básicas de los lenguajes de alto nivel.

Arquitecturas x86 i386 x86-64 amd64

- La arquitectura de Intel x86 de 32 bits comenzó en el año 1985 con el microprocesador 80386 qué más adelante fue renombrado como "i386"
- x86-64 (también conocido como x64, x86_64, AMD64 e Intel 64) es la versión de 64 bits del conjunto de instrucciones x86. La especificación fue creada por AMD
- Linux denomina a la arquitectura x86 como i386 y a la arquitectura x86-64 como amd64.
- En esta asignatura se va a trabajar principalmente con la arquitectura x86 ya que las prácticas se realizan en esta arquitectura de 32 bits.

ISA x86

- El objetivo de este capítulo es tener los conocimientos necesarios para la programación de bajo de nivel (lenguaje ensamblador) de la arquitectura de 32 bits de Intel: x86
- Fases en el proceso de traducción (compilación, ensamblaje, módulos fuente, módulo objeto, módulo ejecutable, enlazado, resolución de referencias, bibliotecas, sistema operativo, etc) del módulo fuente y carga en memoria del módulo ejecutable.
- El programador de bajo de nivel necesita el conocimiento de la ISA, es decir:
 - Registros de propósito general internos de la CPU
 - Registro de estado o registro de flags
 - Direccionamiento de la memoria principal
 - Diferencia entre la memoria interna (registros) y la memoria externa (memoria principal)

ISA x86

- El programador de bajo de nivel necesita el conocimiento de la ISA, es decir:
 - Tipos de datos y su formato: enteros complemento a 2
 - Tamaño de los operandos: sufijos b,w,l,q de los mnemónicos
 - Alineamiento de los datos en la memoria: little endian
 - Modos de direccionamiento: inmediato, directo, etc
 - Sintaxis del lenguaje "AT&T"
 - Directivas del traductor assembler "as"

Ejercicios

- [sum1toN.s](#) [./as_code/sum1toN.s]
 - Programa inicial de referencia
 - sección de datos y sección de instrucciones: directivas .section .data y .section .text
 - direccionamientos: inmediato, a registro y directo
 - estructura de un bloque: saltos incondicionales y condicionales

- etiqueta global
- llamada a funciones del sistema operativo y paso del argumento por registro
- comentarios en lenguaje C y pseudocódigo

Ejercicios

- [sum1toN_v2.s](#) [./as_code/sum1toN_v2.s]
 - Declaración de variables: En la sección de datos reservar memoria para el dato e inicializarlo. Directivas .byte, 2byte, 4byte
 - instrucción de salto condicional: jnz ó jne
- [sum1toN_v3.s](#) [./as_code/sum1toN_v3.s]
 - Directiva .string: reserva de memoria e inicialización con caracteres ASCII
 - llamada a funciones de la librería standard de C como "puts()" y pase de argumentos a través de la pila
- [sum1toN_v4.s](#) [./as_code/sum1toN_v4.s]
 - llamada a la función "exit()" de la librería standard de C

Ejercicio

- [sum1toN_v5.s](#) [./as_code/sum1toN_v5.s]
 - Macros con la directiva .equ
 - Inicializar una array de datos con los diez primeros números enteros impares y sumarlos
- [sum1toN_v6.s](#) [./as_code/sum1toN_v6.s]
 - Macros con la directiva .equ
 - Inicializar una array de datos con los diez primeros números enteros impares y sumarlos
 - Bucle con la instrucción "cmp"
 - [x86](https://www.felixcloutier.com/x86/) [https://www.felixcloutier.com/x86]: [cmp](https://www.felixcloutier.com/x86/cmp) [https://www.felixcloutier.com/x86/jcc] , [jcc](https://www.felixcloutier.com/x86/jcc)
 - Visualizar el resultado con la función printf() de la librería standard de C

Ejercicio

- [sum1toN_v6.s](#) [./as_code/sum1toN_v6.s]
 - Pasar los dos argumentos a través de la pila insertando primero el último argumento printf(1º arg, 2º arg)
- [sum1toN_v7.s](#) [./as_code/sum1toN_v7.s]

```
### Descripción: Macros con la directiva .equ
###           Inicializar una array de datos con los diez primeros números
enteros impares y sumarlos
###           bucle con la instrucción "cmp"
###           Visualizar el resultado con la función printf() de la librería
standard de C
###           Pasar los dos argumentos a través de la pila insertando primero el
```

Subrutinas

Introducción

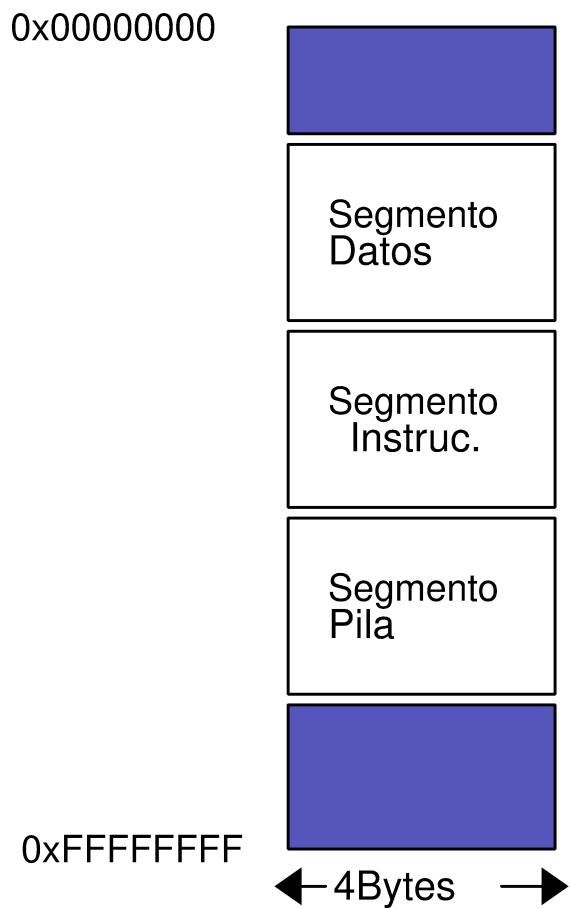
- A las funciones de los lenguajes de alto nivel en el nivel máquina se les llama subrutinas
- Las subrutinas son bloques de código que pueden ser llamadas desde otro bloque para su ejecución y posterior retorno a la sentencia siguiente al punto de llamada.
- A las subrutinas se les pueden pasar parámetros o argumentos a través de unidades de memoria de diferentes formas:
 - En la arquitectura x86 los parámetros se pasan a través de la pila

La pila a nivel máquina x86

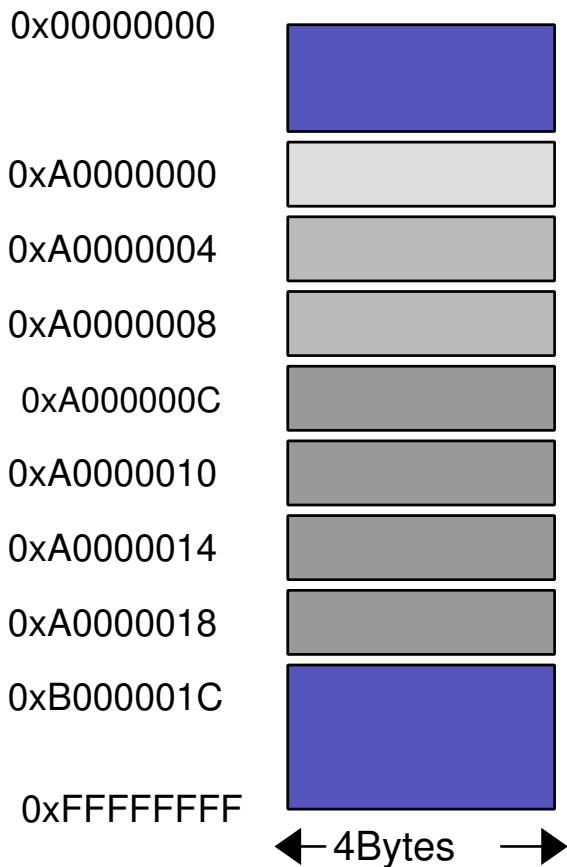
- Es una estructura de datos que se construye en la **memoria principal** y constituye una sección diferente (stack section) de la sección de datos y sección de texto.
- La pila es una 'sección' del programa en ejecución en la memoria principal. A diferencia de la sección de datos y la sección de instrucciones la pila se crea en tiempo de ejecución ,no durante la carga en memoria.
- En la pila los datos se van insertando (apilando) secuencialmente y también se van extrayendo (desapilando)
- Es una estructura de datos tipo LIFO (last input first output): El último elemento apilado es el primero que se puede extraer.
- La instrucción para insertar un dato el **PUSH** y para extraer es **POP**.

```
push %eax # se inserta en la pila y se apila el contenido de EAX  
pop %ebx # se extrae de la pila el último dato apilado y se guarda en EBX  
push etiqueta # ¿Da error? ¿Por qué?
```

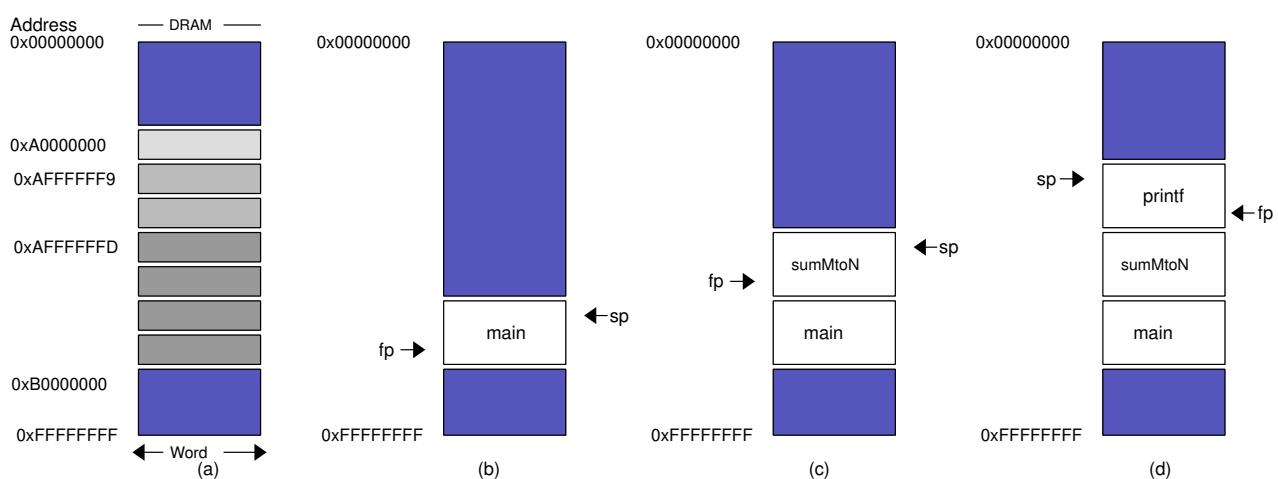
La pila a nivel máquina x86



- [pila](#) [./pila.html]
- La anchura de la pila son 32 bits
- Se apila en el sentido decreciente de las direcciones de memoria



Anidamiento de llamadas a funciones: FRAME



- Cada rutina o subrutina tiene que generar por programa su propio segmento dentro de la sección stack.

Punteros del Frame Activo

- fp: frame pointer y sp: stack pointer
- el puntero fp apunta a la parte baja del frame activo y el sp apunta a la parte alta (último dato apilado) del frame activo

- En la arquitectura x86 el la función fp la realiza el registro **EBP** (Bottom Pointer) y la función sp la realiza el registro **ESP**

Anidamiento de Llamadas a funciones: FRAME

- Un ejemplo típico de anidamiento de funciones son las subrutinas recurrentes o recursivas.
- De aquí en adelante según se van aprendiendo los diferentes conceptos, ponerlos en práctica con [La Animación de Recursión](https://diveintosystems.org/book/C8-IA32/recursion.html) [<https://diveintosystems.org/book/C8-IA32/recursion.html>]: Analizar la función sumr en C y en ASM. Bajar al apartado 8.6.1. Animation: Observing How the Call Stack Changes.
- Ejercicio propuesto al final del capítulo: sum1toN.s como subrutina recursiva

Pase de los argumentos y Llamada a la Subrutina

- función en lenguaje C: sumMtoN(M,N)
- Primero se pasa el último argumento de la subrutina y el último argumento en pasar es el primer argumento de la subrutina
- A continuación se llama a la subrutina
- Código de pase de argumentos y llamada a la subrutina

```
main:
    xxxx xxx
    xxxx xxx
    push N
    push M
    call sumMtoN
    xxxx xxx
    xxxx xxx
```

- Estado de la pila después de insertar los argumentos y antes de ejecutarse la instrucción "call" → ¿?

Conservación de la dirección de retorno

- El procesador durante el ciclo de instrucción de "call" inserta la dirección de retorno en la pila y salta a la dirección de comienzo de la subrutina introduciendo la dirección de la subrutina en el contador de programa.
- Estado de la pila después de ejecutarse el **call** → ¿?

Comienzo de la Subrutina: Prólogo

- Es necesario conservar el fp de la rutina que realiza la llamada para una vez finaliza la subrutina volver a activar el frame de la rutina que llama.

```
sumMtoN:
    push %ebp
```

- El epílogo consisten activar un nuevo frame apuntando los dos punteros sp y fp al primer dato insertado.
- Instrucciones del Prólogo

```
sumMtoN:  
    push %ebp  
    mov %esp,%ebp  
    XXX XXX
```

- Estado de la pila: → ¿?

Variables locales

- Las variables locales se definen en la pila después del prólogo reservando la memoria necesaria. Esto se realiza moviendo el stack pointer tantos bytes como ocupen las variables locales

Recuperación de los Argumentos

- Pasar los argumentos de la pila a un registro para poder ser utilizados. Por ejemplo a los registros ECX y EDX en el caso de tener sólo 2 argumentos.

Fin de la subrutina: Epílogo

- El epílogo consiste en desactivar el frame de la subrutina, activar el frame de la rutina que realizó la llamada y saltar a la dirección de retorno.
- Desactivar el frame de la subrutina: Que el sp apunte a la misma dirección que el bp.
- Activar el Frame de la rutina: Actualizar el bp con el valor anterior al de la subrutina.
- Saltar a la dirección de retorno: El sp ha de estar apuntando a donde se encuentra la dirección de retorno antes de ejecutar la instrucción `ret`
- Código del epílogo:

```
XXX XXX  
mov %ebp,%esp  
push %ebp  
ret
```

- las instrucciones del epílogo son el proceso inverso del prólogo
- Estado de la pila: → ¿?

Valor de retorno

- El valor de retorno no se pasa a través de la pila, se pasa a través del registro EAX

Preservación de los registros

- Es necesario preservar los registros EAX, ECX y EDX antes de llamar a la subrutina: la subrutina podrá utilizarlos libremente
- Es necesario preservar los registros EBX,ESI,EDI,ESP,EBP en caso de ser utilizados por la subrutina. Su valor ha de ser el mismo antes y después de la llamada a la subrutina.

Anidamiento de llamadas a funciones: FRAME RECURRENTE

- Comprender completamente [La Animación de Recursión](https://diveintosystems.org/book/C8-IA32/recursion.html) [<https://diveintosystems.org/book/C8-IA32/recursion.html>]: Analizar la función sumr en C y en ASM. Bajar al apartado 8.6.1. Animation: Observing How the Call Stack Changes.
- Ejercicio: Desarrollar el código "sum1toN.s" como subrutina recursiva

Ejemplo: Subrutina sumMtoN.s

- Código:

```
## Subrutina sumMtoN.s
### Descripción: el shell de linux llama a la función main() y este llama a la
subrutina suma()
    el shell de linux pasa a main() los strings de la línea de comandos
pero aquí no se utilizan
    main() se va a definir como una subrutina (prologo,epilogo,etc ...)
    suma() tiene dos argumentos M y N y calcula la suma de números enteros desde
M hasta N
###
### gcc -m32 -o sumMtoN sumMtoN.s
### Ensamblaje as --32 -o sumMtoN.o sumMtoN.s
### linker -> ld -e main -melf_i386 -I/lib/i386-linux-gnu/ld-linux.so.2 -o sumMtoN
sumMtoN.o

### manual de instrucciones x86 -> https://www.felixcloutier.com/x86/

.section .data
M: .byte 3
N: .byte 10
resultado: .byte 0

## el shell de linux llama a main()
## el shell como funcion llamante ha preservado eax, ecx y edx

.global main
.section .text
main:
## prologo de main
push %ebp
mov %esp,%ebp
## variable local de main
sub $4,%esp
## como funcion que ha sido llamada por el shell preserva ebx, esi y edi para el
shell
    push %edi
    push %esi
    push %ebx
```

```

## capturar argumentos de la funcion llamante (linea de comandos del shell)
## en este caso no utilizo los argumentos
## podría procesar los argumentos del shell y utilizar cualquiera de los registros
eax,ebx,ecx,edx,esi,edi
## en este caso no hay proceso

## Voy a llamar a suma() y empiezan los preparativos
## preservar los registros que puede utilizar suma() libremente
push %edx
push %ecx
push %eax

## pasar los argumentos a suma()
movb N,%ecx ## segundo argumento
    push %ecx
movb M,%edx ## primer argumento
    push %edx

## Llamar a la subrutina
call suma

## Guardar el resultado de suma()
movb %al,resultado

## recuperar los registros salvados antes de la llamada suma()
pop %eax
pop %ecx
pop %edx

## podría procesar los datos y utilizar cualquiera de los registros
eax,ebx,ecx,edx,esi,edi y la variable resultado
## En este caso no proceso nada, tengo el resultado en AL

## Finalizo la funcion main() como toda subrutina retornando a la función
## llamante, en este caso el shell de linux
## como funcion que ha sido llamada por el shell recupera ebx, esi y edi para
el shell
    pop %ebx
    pop %esi
    pop %edi

## Retorna un valor al shell a traves de EAX por CONVENIO
movb resultado,%al

## epilogo de main
mov %ebp,%esp      # frame anterior
pop %ebp

ret

```

```
### Funcion que calcula la suma de numeros de enteros en la secuencia entre dos
limites dados
.type suma, @function
.section .text
suma:
## Pasos iniciales antes del procesamiento de los argumentos de suma()
## prologo: genera un nuevo frame para suma()
push %ebp
movl %esp,%ebp

# una variable local de 4 bytes para suma()
subl $4,%esp

## preserva los registros utilizados por main()
push %edi
push %esi
push %ebx

## captura de argumentos de la función suma()
movb 8(%ebp),%al      #1º argumento
movb 12(%ebp),%cl     #2º argumento

## algoritmo de la subrutina suma(): suma desde el 1º arg hasta el 2º arg
## se pueden utilizar cualquiera de los registros eax,ebx,ecx,edx,esi,edi
## utilizo "al" como suma parcial y "bl" como sumando
movb %al,%bl
bucle:
inc %bl
addb %bl,%al
cmpb %bl,%cl
jg bucle

## el resultado esta en AL por CONVENIO

## Comienza el retorno a la función llamante main()
## recupera los registros salvados para main()
pop %ebx
pop %esi
pop %edi

## epilogo
mov %ebp,%esp          # frame anterior
pop %ebp
ret                   # recuperar dirección de retorno

.end
```

Ejercicio

- [sumMtoN_preservar.s](#) [./as_code/sumMtoN_preservar.s]
 - Convertir el bloque sumMtoN en una subrutina
- [recursion.c](#) [./as_code/recursion.c]
 - Convertir la función recursion.c en una subrutina recursiva

LLamadas al Sistema

Funciones del sistema

- Privilegios del sistema, no del usuario: acceso al hardware (por ejemplo acceder a la pantalla para imprimir una imagen , acceder al disco para escribir un fichero, arrancar o finalizar un proceso, etc ...)
- Volcar el contenido de la cabecera "/usr/include/asm/instd_32.h"

```
exit 1
fork 2r
open 5
close 6
```

LLamada y pase de los argumentos

- El código de identificación de la función se pasa a través del registro EAX
- Los argumentos de la función se pasan a través de los registros EBX,ECX,EDX,ESI,EDI,EBP y en ese orden.
- La llamada se realiza con la instrucción **int \$0x80**
- Ejemplo con el servicio "exit":

```
mov $1,%eax
mov $0,%ebx
int $0x80
```

- Ejercicio: código ensamblador con servicio "write" según el prototipo del manual "man 2 write"

Ejercicios Propuestos Temas 1-6

- En los apuntes [on line](#) [https://kandido.github.io/eecc_book.html], en la sección III de "Ejercicios de Teoría" se invita a realizar como mínimo los ejercicios especificados en la "Lista mínima de ejercicios" de los cuales en clase se realizarán los siguientes:
 - Arquitectura de Von Neumann: 1.1,1.2,1.3,1.4 y 1.6 (no programar en lenguaje ensamblador IASSim)
 - Representación de los Datos: 2,3,4 y 5
 - Programación en Lenguaje ensamblador AT&T: 1.1, 1.2, 2.2.

Ejercicios de Prácticas I

- Datos:
 - Todos los números enteros de todos los ejercicios tendrán un tamaño de 2 bytes.
 - Dado el string "The United Nations Convention on the Rights of the Child is an important agreement by countries who have promised to protect children's rights." contar el número de veces que aparece la letra "a"
 - Dado el array de números enteros: 0xFFFF,0x03,0x7FA0,0x87BC,0xBA34,0xFA,0x732 buscar el número con el valor negativo de menor módulo
 - Dado el array lista_A : 12,-33,45 y el array lista_B: 76,89,-65 generar el array lista_D suma de los elementos de lista_A y lista_B
 - Dado el array lista_A : 12,-33,45 y el array lista_B: 76,89,-65 generar el array lista_C formado el encadenamiento de lista_A y lista_B
- Llamadas al Sistema
 - Imprimir en la pantalla el mensaje "Hola Mundo" mediante un programa en asm at&t:
 - Mediante las funciones de C printf y puts según los prototipos de los manuales "man 3 printf" y "man 3 puts"
 - Mediante la función del sistema "write" según el prototipo del manual "man 2 write"

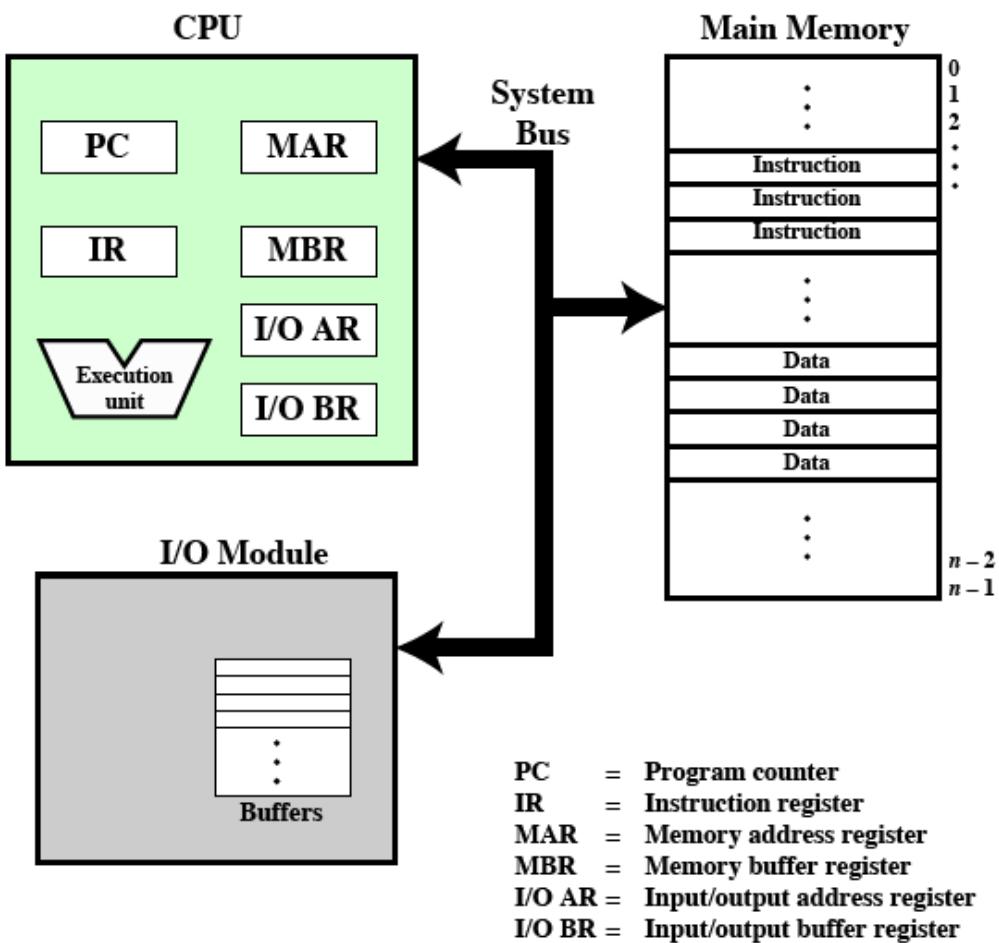
Tema 3 Representación de los Datos: Números Reales

Tema 7 Unidad Central de Proceso

- Introducción
- Fases de ejecución de una instrucción
- Ruta de datos
- Segmentación
- Arquitecturas CISC, RISC y VLIW
- Conjunto de instrucciones

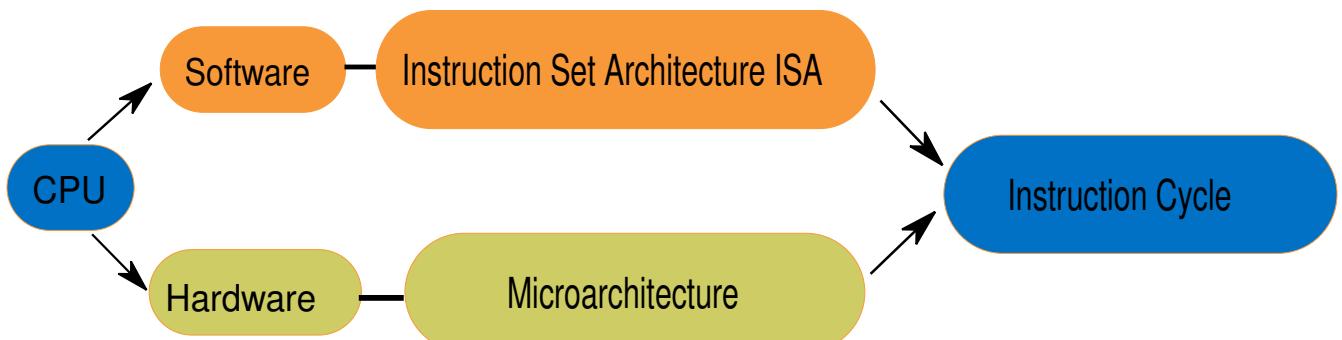
Introducción

- El objetivo principal de la CPU es la implementación del ciclo de instrucción. Es el soporte hardware para poder llevar a cabo todas las operaciones que conllevan las instrucciones de un programa.
- Nombres: Procesador, microprocesador (El componente electrónico básico es el transistor con un tamaño en sus orígenes del orden de la micra), ..
- Central porque la computadora tiene varios procesadores: Por ejemplo el controlador de la memoria y los controladores de los periféricos.
- La CPU es una de las unidades básicas que conforman la arquitectura Von-Neumann (CPU-MP-IO) y Buses.



Logo 1. IAS_Architecture

Puntos de Vista: SW vs HW



Logo 2. Puntos de Vista: SW y HW

Puntos de Vista: SW vs HW

- La CPU se puede ver desde el punto de vista del **programador** o desde el punto de vista del diseñador de procesadores (microarquitectura)
- Desde el punto de vista del programador interesa conocer:
 - La Arquitectura del Repertorio de Instrucciones (ISA): Formatos de datos e instrucciones, operaciones, modos de direccionamiento, registros, direccionamiento de la memoria, alineamiento de los datos, etc
 - Modos de Funcionamiento de la CPU: modo superusuario, modo usuario, modo interrupción.

- Desde el punto de vista de la Microarquitectura u organización interna de la CPU.
 - Las unidades básicas de la CPU son: Unidad de Control (UC), y la Ruta de Datos (ALU,FPU,MMU,Registros y circuitos de enrutamiento como multiplexores, conmutadores, etc).
 - La Unidad de Control se encarga de la gestión de las operaciones a realizar, mediante microórdenes, para completar el ciclo de instrucción. Electrónicamente es un circuito digital secuenciador cuyas secuencias son las fases del ciclo de instrucción.
 - Las microórdenes son señales digitales binarias.

Fases de ejecución de una instrucción

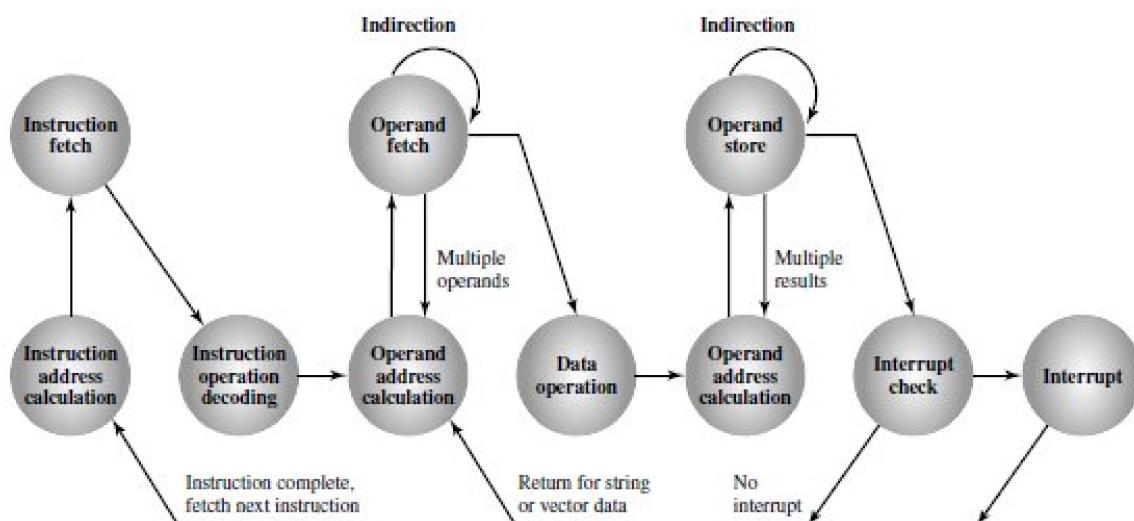


Figure 12.5 Instruction Cycle State Diagram

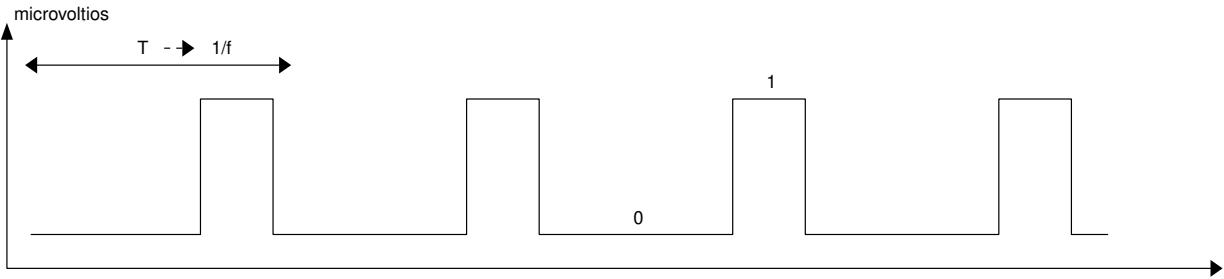
Logo 3. Diagrama de Estados del Ciclo de Instrucción

Instruction and **O**perand

I-Fetch / I-Decode / O-Fetch / I-Execute / O-Write / I-Interrupt / Next-I

Sincronismo de las operaciones: señal de reloj

- Señal de Reloj: Señal binaria (niveles 0 y 1) periódica (cada ciclo de la señal se repite)
- Físicamente es una señal electrónica: puede ser una tensión (microvoltios) ó una corriente (microamperios)



- Flancos de reloj: Cambio de nivel 0→1 (positivos) o 1→0 (negativos)
- El reloj se utiliza para poder sincronizar las operaciones de los circuitos electrónicos
 - Pej : instantes en que cambian las salidas de un registro
 - Pej : instantes en que cambian las salidas de la memoria
 - Pej : instantes en que cambian las microórdenes de la unidad de control
- El objetivo es facilitar el diseño de los circuitos electrónicos digitales con que se implementan

Unidad de Control y Ruta de datos

- Primero diseñaremos la Ruta de Datos de cada Instrucción y finalmente la Unidad de Control que gestiona la ruta de datos de cada instrucción
- Es necesario diseñar una ruta de datos diferente para cada instrucción y una única Unidad de Control para todas las instrucciones

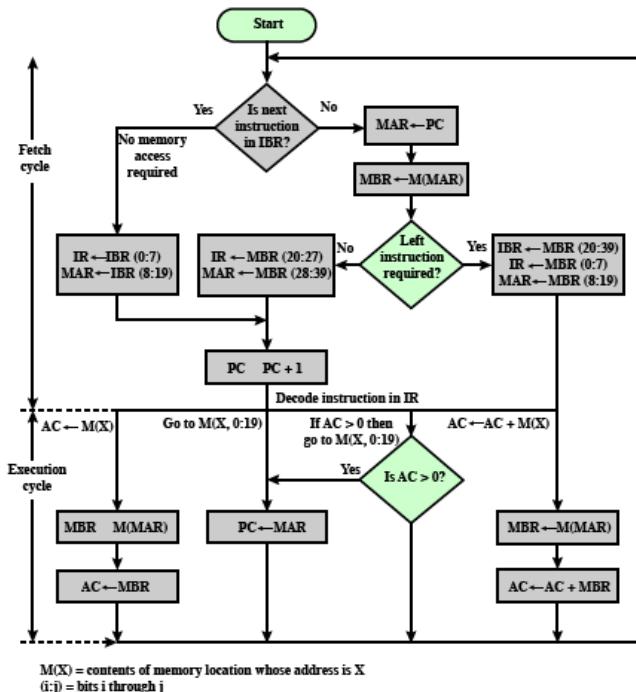
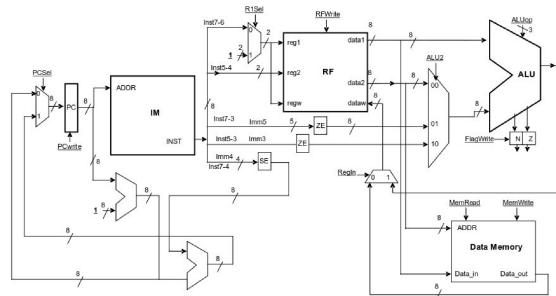


Figure 2.4 Partial Flowchart of IAS Operation

Logo 4. IAS Operation

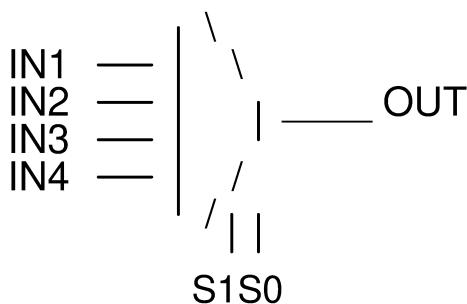
Rutas de Datos



Logo 5. Data Path Single Cycle

Rutas de Datos

- Los bloques que forman parte de la ruta de datos se encuentran interconectados uno a uno mediante hilos (pistas o cables)
- En la imagen se distinguen los hilos de la ruta de datos de los nombres de las microoperaciones (subrayados)
- Inst7-6 : son los bits 7-6 del campo de operación para cada instrucción → Dependiendo del valor de estos dos bits la ruta de datos cambia
- InstM-N: son la secuencia de bits M(M-1) hasta N del campo de operaciones para cada instrucción. Mismo concepto que para Inst7-6
- Función del multiplexor:
- se conecta sólo una de las 4 entradas a la salida. La entrada INx se selecciona con S1S0



Unidad de Control: Microoperaciones y Ruta de Datos

- La Unidad de Control gobierna la ruta de datos para completar todas las fases del ciclo de instrucción de cada instrucción ISA
- Microoperaciones: operaciones realizadas por la CPU internamente, al ejecutar cada fase del ciclo de instrucción de una Instrucción Máquina.

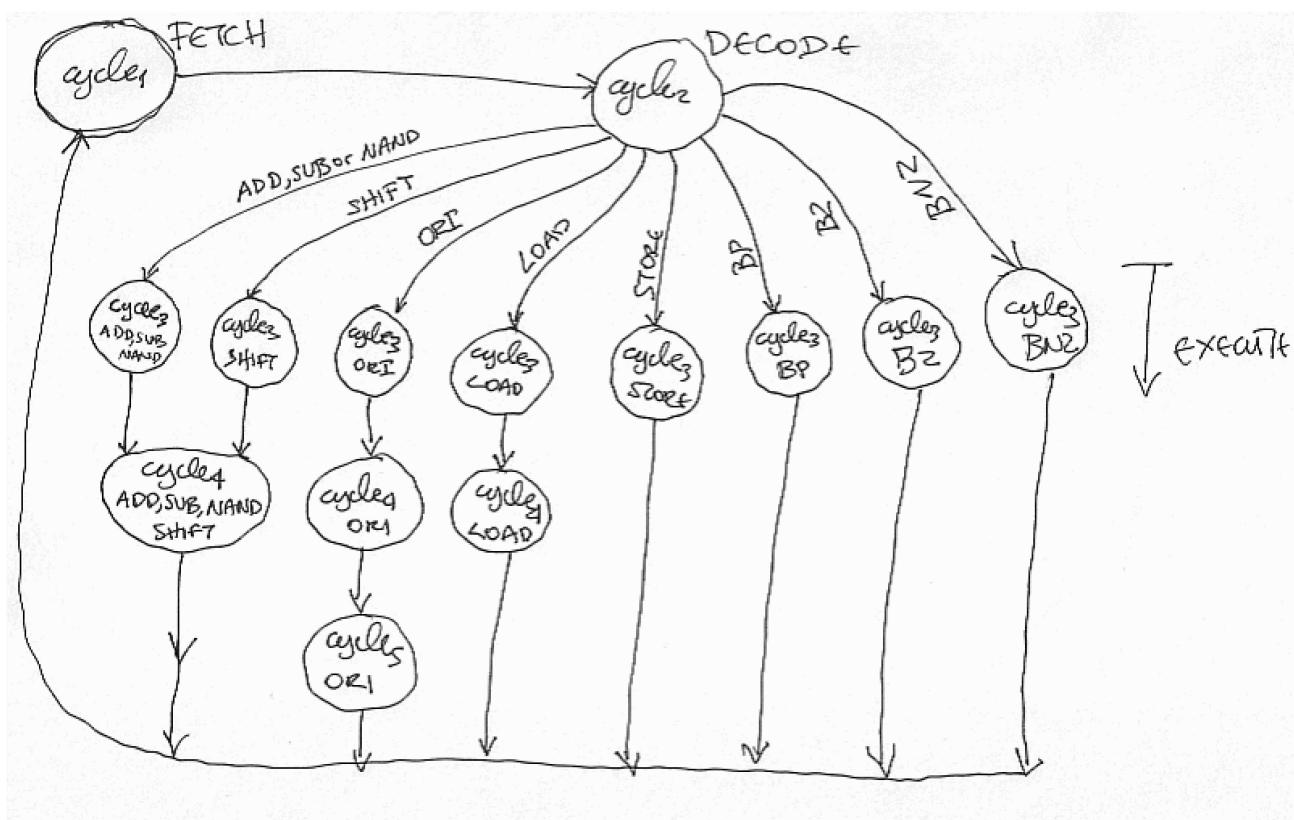
- Ejemplos de microoperaciones: escribir en el registro MAR, orden de lectura a la MPrincipal, leer de MBR, interpretar de IR, incrementar PC, etc
- Microoperaciones síncronas con el flanco positivo del reloj del secuenciador (unidad de control) de la cpu

Unidad de Control: Secuenciador o Máquina de Estados Finitos

INSTRUCTION	Inputs			Outputs									
	INST0-3	N	Z	PCSel	PCWrite	RegWrite	MemRead	R1Sel	MemWrite	ALU2	RFin	ALUop	
LOAD	0000	X	X	0	1	1	1	0	0	X	1	XXX	
STORE	0010	X	X	0	1	0	0	0	1	X	X	XXX	
ADD	0100	X	X	0	1	1	0	0	0	00	0	000	
SUB	0110	X	X	0	1	1	0	0	0	00	0	001	
NAND	1000	X	X	0	1	1	0	0	0	00	0	011	
ORI	X111	X	X	0	1	1	0	1	0	01	0	010	
SHIFT	X011	X	X	0	1	1	0	0	0	10	0	100	
BZ	0101	X	0	0	1	0	0	X	0	X	X	XXX	
	0101	X	1	1	1	0	0	X	0	X	X	XXX	
BNZ	1001	X	0	1	1	0	0	X	0	X	X	XXX	
	1001	X	1	0	1	0	0	X	0	X	X	XXX	
BPZ	1101	0	X	1	1	0	0	X	0	X	X	XXX	
	1101	1	X	0	1	0	0	X	0	X	X	XXX	

- El circuito electrónico controlador es un secuenciador digital síncrono o máquina de estados finitos
- La tabla presenta las entradas y salidas del secuenciador para cada instrucción
- Las entradas del secuenciador son: el código del campo de operaciones de la instrucción, el código del registro de flags, la etapa del ciclo de instrucción y el RELOJ.
- La salida es el conjunto de microoperaciones para esa entrada que cambiará sincronamente con el RELOJ.

Unidad de Control: Secuenciador o Máquina de Estados Finitos



- Cada instrucción tiene su propia secuencia
- En cada fase será necesario ejecutar las microoperaciones para completar dicha fase
- La unidad de control se diseña para que cada fase se complete en un ciclo de reloj

Unidad de control: micro-operaciones

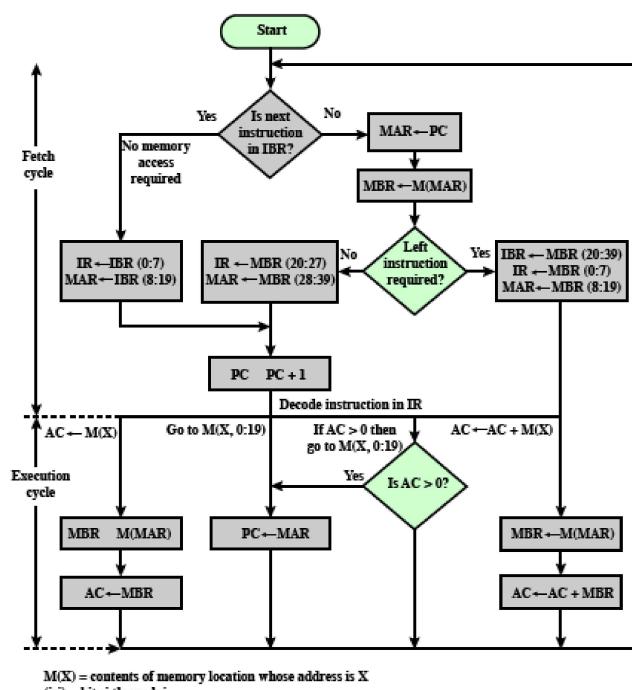


Figure 2.4 Partial Flowchart of IAS Operation

Logo 6. IAS Operation

Microarquitecturas: RISC y CISC

- CISC: Complex Instruction Set Computer
 - Intel
 - elevado número de códigos de operación
 - instrucciones máquina que realizan operaciones "complejas" y múltiples modos de direccionamiento complejos
 - Ventajas: fácil de compilar, módulo binario reducido
 - Inconvenientes: Unidad de Control compleja, alto consumo y elevado coste
 - Aplicación : servidores
- RISC: Reduced Instruction Set Computer
 - ARM
 - reducido número de códigos de operación
 - operaciones "sencillas" y modos de direccionamiento sencillos
 - Ventajas: unidad de control sencilla, bajo consumo y bajo coste
 - Inconvenientes: no puede competir en número de procesos a ejecutar y a alta velocidad,
 - Aplicación: computación portátil

Microarquitectura Segmentada

- Pipeline → tuberías
- Ejemplo de la cadena para la limpieza automática de coches
- Múltiples fases del ciclo de instrucción ejecutándose simultáneamente: paralelismo
- Calcular el Throughput: instrucciones ejecutadas por unidad de tiempo

Arquitectura VLIW vs Superscalar

- VLIW: Very Large Instruction Word
- Superscalar
- Múltiples Unidades de Ejecución: ¿Cómo se distribuyen las instrucciones para cada Unidad de Ejecución?

Investigar la microarquitectura del ordenador personal

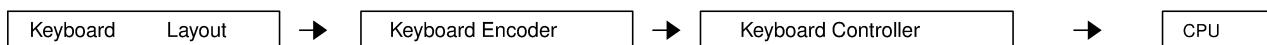
- Herramientas

Tema 8: Mecanismos de Entrada/Salida

Periféricos

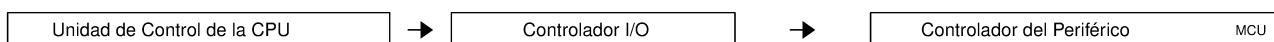
- Modelo:
 - Media: magnético, óptico, electrónico, etc
 - Driver HW : interfaz electrónica con ... brazo mecánico, láser , ink, altavoz, monitor pixel, etc

- [Hard Disk](http://theithollow.com/2013/11/disk-latency-concepts/) [http://theithollow.com/2013/11/disk-latency-concepts/]
- Controlador del Periférico : Secuenciador para ¿? Ej [Disk Buffer](https://en.wikipedia.org/wiki/Disk_buffer) [https://en.wikipedia.org/wiki/Disk_buffer]
- Firmware → software para ¿?, comandos, programas, lenguajes (SCSI, SATA, etc), ...

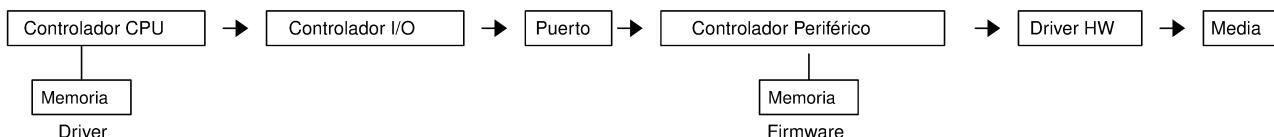


Arquitectura Von Neumann

- Controladores E/S
- Intel: Chipset
- Comunicación entre el Controlador E/S de la CPU y el Controlador del Periférico
 - Modelo BUSES CPU-Memoria Principal
 - Protocolo de Comunicaciones (buses), Memoria de acceso E/S (Puertos de Entrada/Salida)

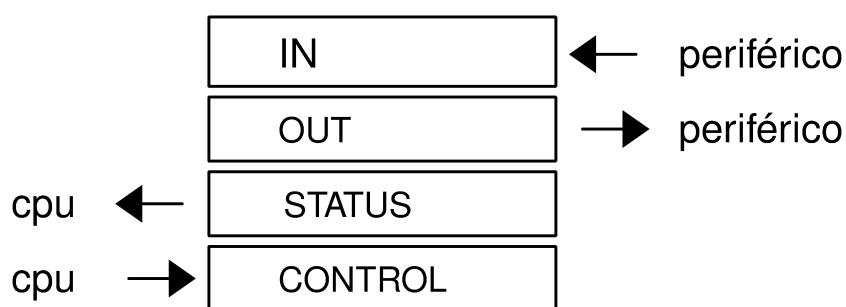


Controlador I/O



Puertos

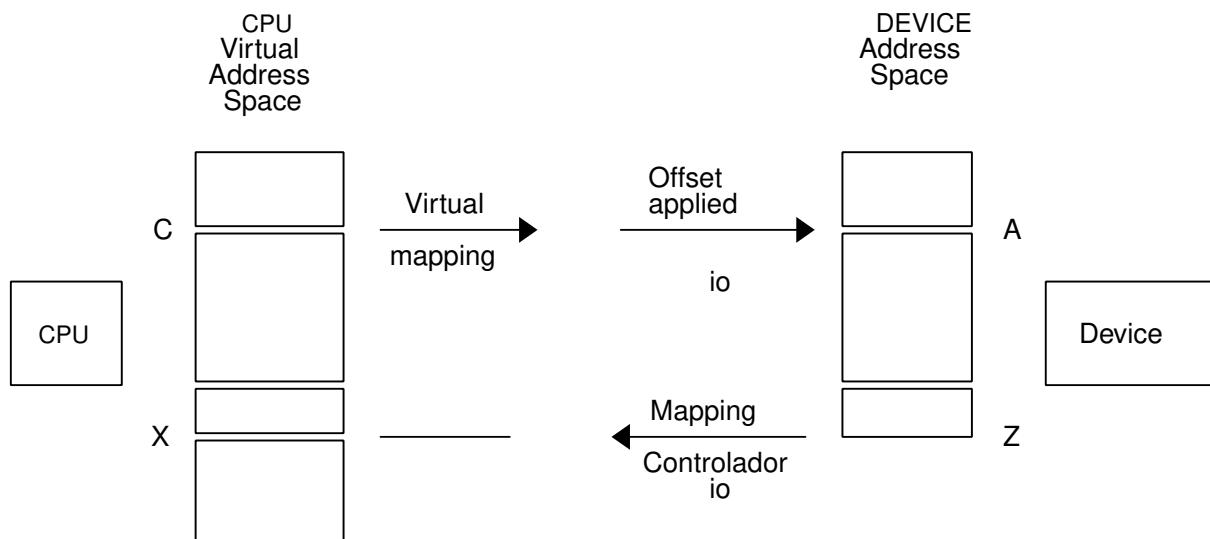
controlador i/o
puerto



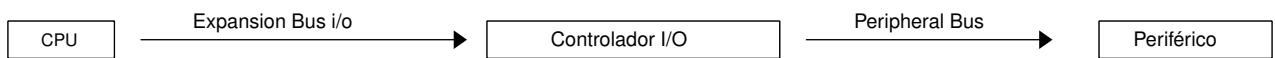
- Espacio de direcciones:
 - **MMIO**

- PMIO

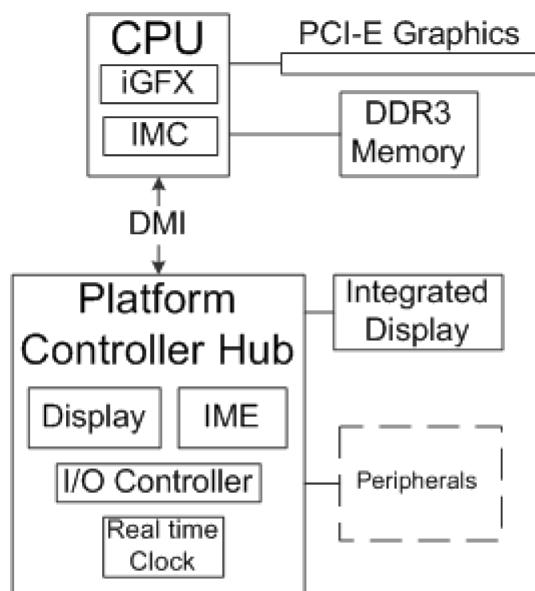
Espacio de Direcciones



Buses



- Buses
 - del sistema : memoria e i/o
 - local: PCI
 - expansión: EISA
 - periférico: USB, HDMI,



Logo 7. Lenovo T520

Nivel de Integración

- Evolución de Intel
 - Puentes Norte y Sur
 - La CPU integra el controlador de memoria integrado (IMC) y controladores i/o de video (PCI-E Graphics)
 - El chip PCH es un hub de controladores i/o

Programa de entrada/salida

- Programmed i/o (PIO): Las transferencias de datos mediante mecanismos de E/S por consulta la realiza un programa i/o (PIO) que ejecuta la CPU. La CPU en cada transferencia de datos entre la memoria y el periférico debe de ESPERAR a que dicha transferencia termine.
- Transferencia de 512 bytes entre el puerto 0x380 y un buffer.

```

mov %bx,buf ; destination address. BX es un puntero a un buffer
mov $512,%si ; count. Número de bytes a transferir
mov $0x380,%dx ; source port. DX es un puntero al puerto
loop:
  in %dx,%al ; get byte from i/o port. AL<-DX
  mov %al,(%bx) ; store in buffer      M[bx]<-AL
  inc %bx ; next memory location in buf
  dec %si ; decrement bytes left
  jnz loop
  
```

- ISA:
 - IN: leer un dato del puerto
 - OUT: escribir un dato en el puerto

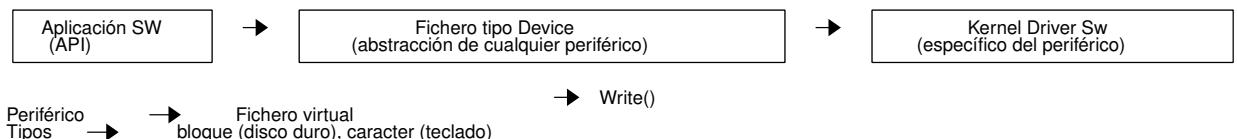
Espacio usuario-kernel-periférico

- Ejemplo: Escritura de un fichero en el disco duro
 - write → syscall → OUT → comando propio del HD



Sistema Operativo

- Las funciones open, write, read, close interactan con el sistema de ficheros virtual



Mecanismos de Implementación de la Interfaz E/S

- All data manipulation not directly performed in the CPU or between CPU and primary memory is I/O.
- PIO:Polling
- Interruption
- DMA: Direct Memory Access

Sincronización por Encuesta

- Polling: encuesta
- Query: encuesta
- Mecanismo
 - Comprobación del estado o 'encuesta-polling'
 - La **CPU consulta** el registro de estado de cada **puerto** al que están conectados los periféricos. Comprueba si algún periférico requiere el servicio de la CPU. Reserves a register for each I/O device. Each register is continually polled to detect data arrival.
 - Es necesario ejecutar programas de atención al periférico cuando este lo requiera: **Sincronización**. Sincronizar significa especificar o indicar cuando se va a realizar la transferencia entre el host(anfitrión) y el periférico.
- CPU
 - Ejecuta el **programa i/o**: un programa que controla DIRECTAMENTE la operación E/S: Programmed I/O → PIO
 - realiza las **transferencias** entre la memoria principal y el controlador i/o
 - '**espera**' al periférico hasta que termine. La CPU espera hasta que concluya la operación E/S.

Sincronización por Interrupción

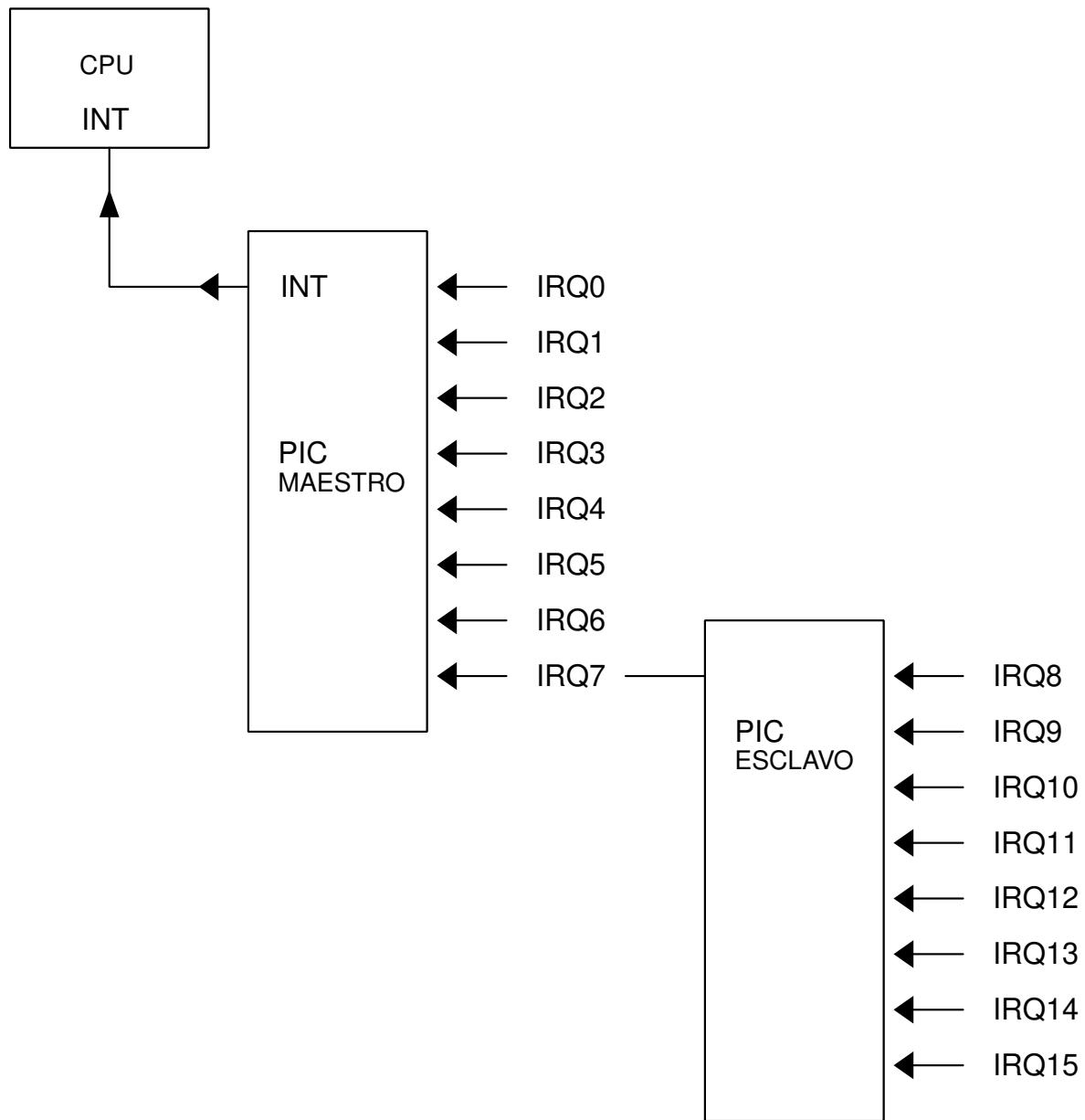
- Interrupt-Driven I/O (Mecanismo de E/S por Interrupción)
- Estructura
 - CPU:
 - Ejecuta el programa de atención a la interrupción. Un programa que controla DIRECTAMENTE la operación E/S.
 - realiza las transferencias entre la memoria principal y el controlador i/o
 - **no espera** al periférico hasta que termine. Es 'interrumpido' cada vez que es necesario realizar una transferencia
 - Al finalizar el ciclo de instrucción de cada instrucción que ejecuta la CPU, se comprueba si la señal de petición de interrupción está activada.

Estructura del Mecanismo de Interrupción

- El periférico a través de una línea eléctrica de entrada de la CPU, solicita al controlador i/o los servicios del kernel
 - El kernel va a ser **INTERRUMPIDO**
- La CPU tiene dos líneas de interrupción:
 - Línea Interrupt ReQuest '(IRQ)' : Maskable
 - Línea Non Maskable Interrupt '(NMI)'
 - La CPU en el ciclo de instrucción tiene en su última fase la fase de Chequeo de Interrupción
 - Si se está solicitando un servicio al kernel, la CPU entra en modo atención a la interrupción y pasa el control al módulo de **Gestión de Interrupciones** del Kernel
- Las líneas de los periféricos para solicitar la interrupción se denominan '(IRQ)'.

Sincronización por Interrupción: Programmable Interrupt Controller

- Las líneas de interrupción de los periféricos (IRQ) se conectan y son controladas por el dispositivo "Programmable Interrupt Controller"



- **Acciones** que realiza el controlador PIC
 - Chequea si se activa alguna señal (Monitorización). En caso de activarse más de una se dar prioridad a la de menor nivel y procede:
 - Convierte la línea activada IRQn en un **vector** (0x00-0xFF)
 - Escribe en el puerto i/o del PIC el vector. El puerto es accesible por parte de la CPU. El **apunta a** la entrada de una **tabla** que contiene un puntero a la rutina de atención a la interrupción (**ISR**)
 - Activa la señal INTR de la CPU
 - Si la CPU lee el valor del vector se desactiva la señal INTR
- A la línea IRQn le corresponde, por defecto según Intel, el vector 'n+32'. Este mapeo lo puede alterar el Sistema Operativo programando el PIC.
 - A la línea IRQ0 le corresponde el vector 0x20
- Cada línea IRQ se puede desautorizar por programa pero dicha interrupción no se pierde.

- Mediante la instrucción cli se hace un clear del flag IF del registro EFLAGS, ignorando la CPU todas las interrupciones hardware.
 - Mediante la instrucción sli hacemos un set de IF.

Tabla del Vector de Interrupciones

- Tabla IVT: Modo Real (antes de cargar el Sistema Operativo)

Interrupt Address	Type	Description
00h	0000:0000h	Processor Divide by zero
01h	0000:0004h	Processor Single step
02h	0000:0008h	Processor Non maskable interrupt (NMI)
03h	0000:000Ch	Processor Breakpoint
04h	0000:0010h	Processor Arithmetic overflow
05h	0000:0014h	Software Print screen
06h	0000:0018h	Processor Invalid op code
07h	0000:001Ch	Processor Coprocessor not available
08h	0000:0020h	Hardware System timer service routine
09h	0000:0024h	Hardware Keyboard device service routine
0Ah	0000:0028h	Hardware Cascade from 2nd programmable interrupt controller
0Bh	0000:002Ch	Hardware Serial port service - COM port 2
0Ch	0000:0030h	Hardware Serial port service - COM port 1
0Dh	0000:0034h	Hardware Parallel printer service - LPT 2
0Eh	0000:0038h	Hardware Floppy disk service
0Fh	0000:003Ch	Hardware Parallel printer service - LPT 1
10h	0000:0040h	Software Video service routine
11h	0000:0044h	Software Equipment list service routine
12h	0000:0048H	Software Memory size service routine
13h	0000:004Ch	Software Hard disk drive service
14h	0000:0050h	Software Serial communications service routines
15h	0000:0054h	Software System services support routines
16h	0000:0058h	Software Keyboard support service routines
17h	0000:005Ch	Software Parallel printer support services
18h	0000:0060h	Software Load and run ROM BASIC
19h	0000:0064h	Software DOS loading routine
1Ah	0000:0068h	Software Real time clock service routines
1Bh	0000:006Ch	Software CRTL - BREAK service routines
1Ch	0000:0070h	Software User timer service routine
1Dh	00000074h	Software Video control parameter table
1Eh	0000:0078h	Software Floppy disk parameter routine
1Fh	0000:007Ch	Software Video graphics character routine
20h-3Fh	0000:0080f-0000:00FcH	SW DOS interrupt points
40h	0000:0100h	Software Floppy disk revector routine
41h	0000:0104h	Software hard disk drive C: parameter table
42h	0000:0108h	Software EGA default video driver
43h	0000:010Ch	Software Video graphics characters
44h	0000:0110h	Software Novel Netware API
45h	0000:0114h	Software Not used
46h	0000:0118h	Software Hard disk drive D: parameter table

47h	0000:011Ch -	Software	Not used
48h	Software	Not used	
49h	0000:0124h	Software	Not used
4Ah	0000:0128h	Software	User alarm
4Bh-63h	0000:012Ch -	Software	Not used
64h	Software	Novel Netware IPX	
65h-66h	Software	Not used	
67h	Software	EMS support routines	
68h-6Fh	0000:01BCh	Software	Not used
70h	0000:01c0h	Hardware	Real time clock
71h	0000:01C4h	Hardware	Redirect interrupt cascade
72h-74h	0000:01C8h - 0000:01D0h	Hardware	Reserved - Do not use
75h	0000:01D4h	Hardware	Math coprocessor exception
76h	0000:01D8h	Hardware	Hard disk support
77h	0000:01DCCh	Hardware	Suspend request
78h-79h	0000:01E0h -	Hardware	Not used
7Ah	Software	Novell Netware API	
78h-FFh	0000:03FCCh	Software	Not used

- Tipos de interrupciones
 - The first 32 vectors are reserved for the processor's internal 'exceptions' (0x00-0x1F)
 - Las interrupciones 0x20-0xFF son interrupciones 'hardware' IRQ.
 - PIC
 - El controlador PIC es el encargado de mapear la señal IRQ a un vector de entrada a la tabla.
 - Periférico IRQ0 → PIC vector 0x20 → Tabla IVT puntero 0000:0080f (RAM) → El puntero contiene la dirección de la función ISR de atención al periférico IRQ0 (RAM). Si esa dirección se carga en el PC, se llama a la función ISR.

Tabla IDT: Modo Protegido

- Modo protegido por el sistema operativo. El S.O. configura la tabla de descripción de interrupciones IDT

IDT Offset	INT #	Description
0x0000	0x00	Divide by 0
0x0004	0x01	Reserved
0x0008	0x02	NMI Interrupt
0x000C	0x03	Breakpoint (INT3)
0x0010	0x04	Overflow (INTO)
0x0014	0x05	Bounds range exceeded (BOUND)
0x0018	0x06	Invalid opcode (UD2)
0x001C	0x07	Device not available (WAIT/FWAIT)
0x0020	0x08	Double fault
0x0024	0x09	Coprocessor segment overrun
0x0028	0x0A	Invalid TSS
0x002C	0x0B	Segment not present
0x0030	0x0C	Stack segment fault
0x0034	0x0D	General protection fault
0x0038	0x0E	Page fault
0x003C	0x0F	Reserved
0x0040	0x10	x87 FPU error
0x0044	0x11	Alignment check
0x0048	0x12	Machine check
0x004C	0x13	SIMD Floating Point Exception
0x00xx	0x14-0x1F	Reserved
0x0xxx	0x20-0xFF	User definable

→ IRQ, SW interrupts

- El contenido depende del kernel del S.O.
- Primera columna: offset a la rutina de atención a interrupción ISR
- Segunda columna: número del vector de interrupción.
- tipos de interrupción
 - 0-0x1F: 'exceptions ERROR' y NMI
 - 0x20-0x2F: INT maskable: IRQ0-----IRQ15
 - 0x30-0xFF: 'exceptions SW'
 - 0x80 → LLAMADAS AL SISTEMA
 - isa x86-64: 'syscall'
 - isa x86: 'int \$0x80'
- ¿A que rutina apunta el vector 0xE? → Page Fault

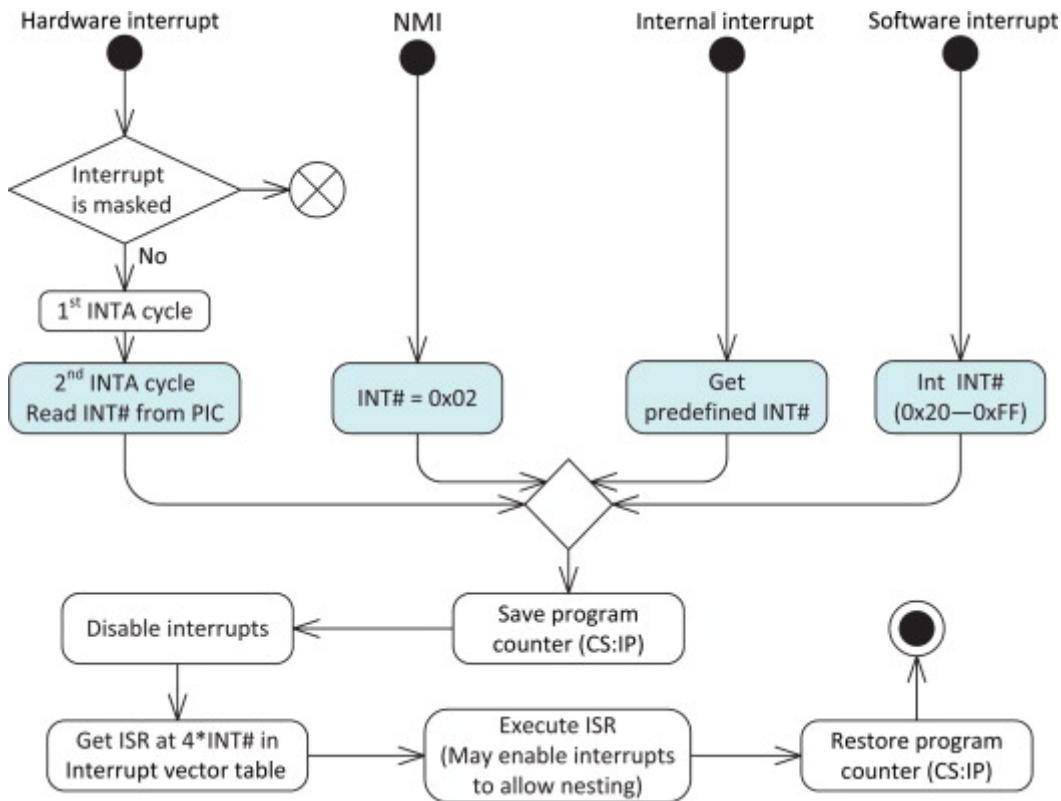
IRQ

IRQ	Interrupt Hardware Device (vector de la tabla)
0	32 Timer
1	33 Keyboard
2	34 PIC Cascade
3	35 Second Serial Port (COM2)
4	36 First Serial Port (COM 1)
5	<Free>
6	38 Floppy Disk
7	<Free>
8	40 System Clock
9	<Free>
10	42 Network Interface Card(NIC)
11	43 USB Port, and Sound Card

- 12 44 Mouse (PS2)
- 13 45 Math Co-Processor
- 14 46 IDE Channel 1
- 15 47 IDE Channel 2

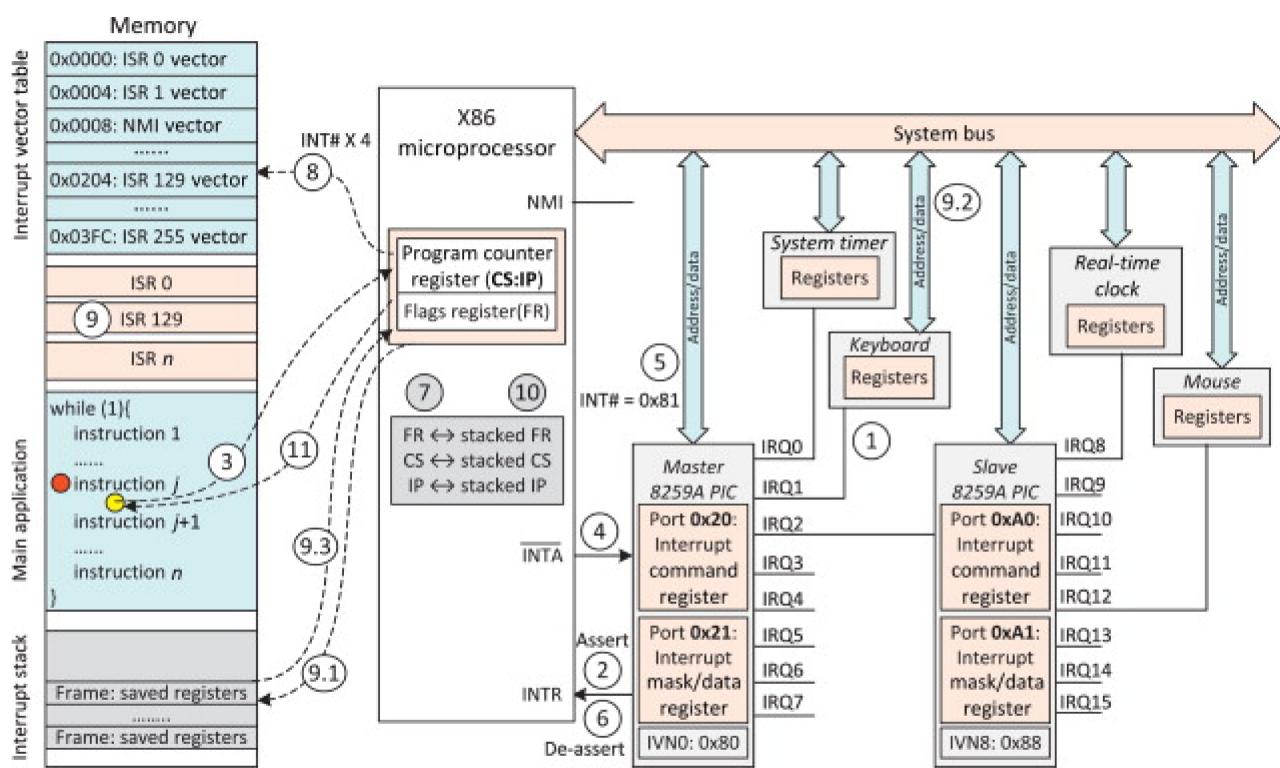
Note: Linux* requires IRQ 0, 2, and 13 to be as shown.

Proceso de Atención a las Interrupciones



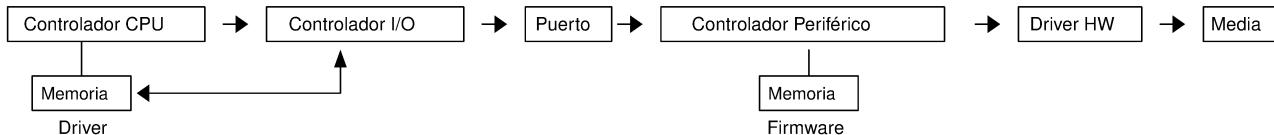
Logo 8. Flujo de Operaciones

Ejemplo: Interrupción realizada al pulsar una tecla

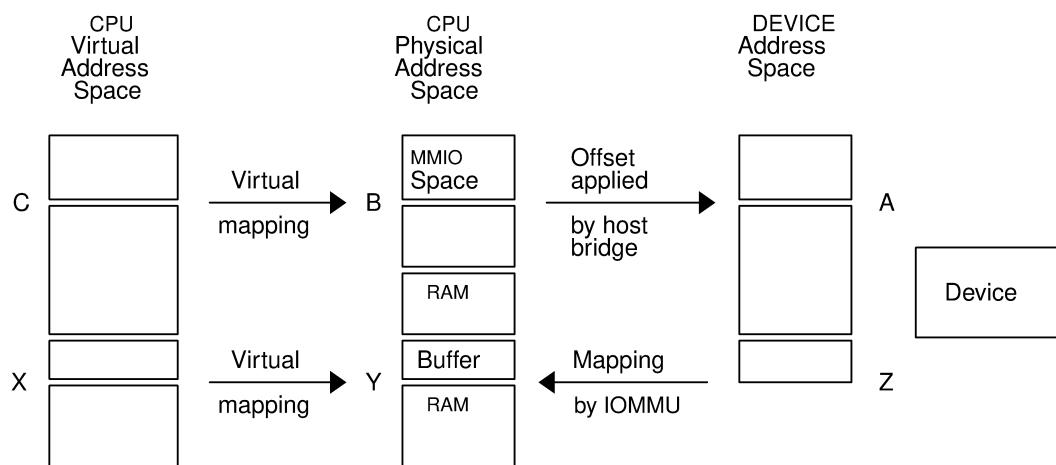


Logo 9. Interrupción del Teclado

DMA



- Mapeo de direcciones
- Hará falta una unidad hardware de traducción del espacio de direcciones del DISPOSITIVO a direcciones físicas de la Memoria Principal : **iommu**



- MMIO: Memory Mapped IO : cpu↔DMA

- Buffer: io transfer data controlled by DMA firmware

Tema 9: Unidad de Memoria

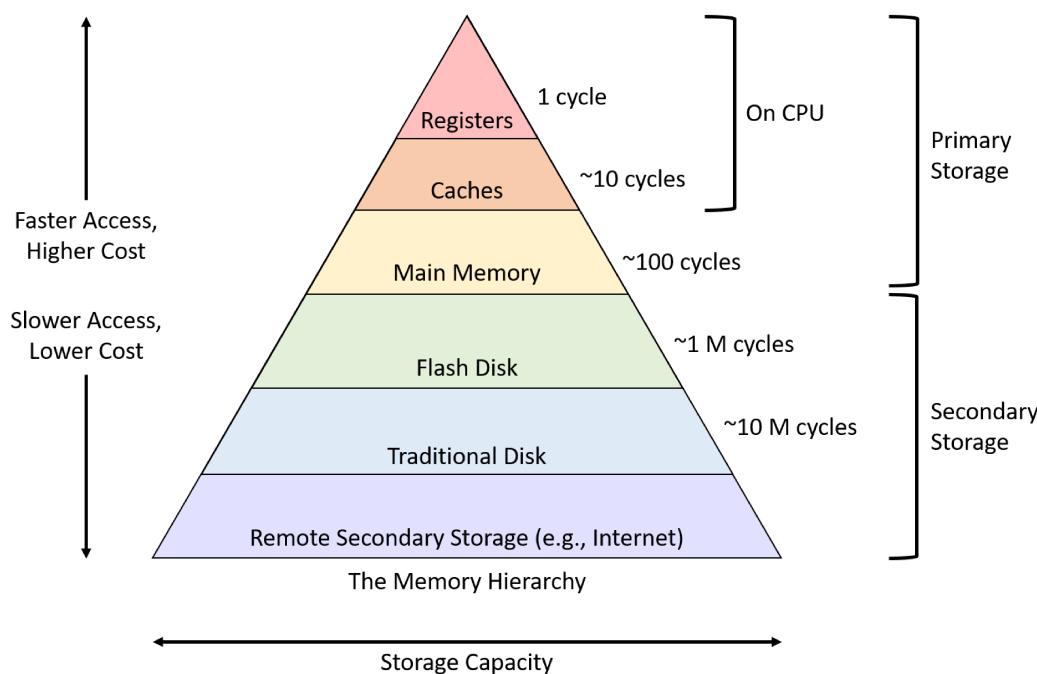
Introducción

- ¿Qué hemos visto relacionado con la Memoria de una Computadora ?
- John von Neumann 1950 : Computadora con programa almacenado
 - Unidad de Memoria: programación binaria → Instrucciones máquina y datos
 - Arquitectura von Neumann: CPU+Unidad de Memoria+I/O+buses
- Memoria: Registros de la CPU y Memoria Principal (Externa a la CPU)
- Ciclo de Instrucción: Captura de Instrucciones y Operandos de la Memoria Principal
- Objetivo de la Memoria Principal:
 - Aumentar la Capacidad de Almacenamiento: En las computadoras multiproceso se ejecutan simultáneamente múltiples programas (centenares) que han de estar accesibles a la CPU.
 - Reducir la Latencia (tiempo de acceso o captura): Hay instrucciones que la cpu ejecuta en un ciclo de reloj (del orden de 1 ns) y el tiempo de captura de la instrucción o dato es del orden de 100 ciclos de reloj.

Perspectivas de estudio

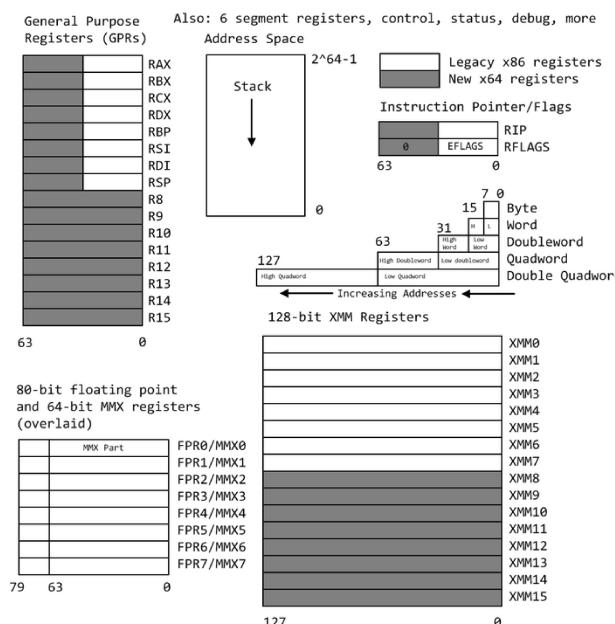
- La memoria es estudiada desde puntos de vista diferentes dependiendo del entorno de estudio.
- Sistemas Operativos: Gestor de Memoria Virtual
- Compilador, Linker,...: secciones de memoria (datos, instrucciones, pila, ...), resolución de direcciones de etiquetas, etc
- ISA: etiquetas, modos de direccionamiento, tamaño de los datos, instrucciones de movimiento de bloques de datos, directivas, etc
- Programador: Variables ordinarias, arrays, estructuras de datos, etc
- Hardware: Organización y Estructura de la Memoria

Jerarquía de Memoria

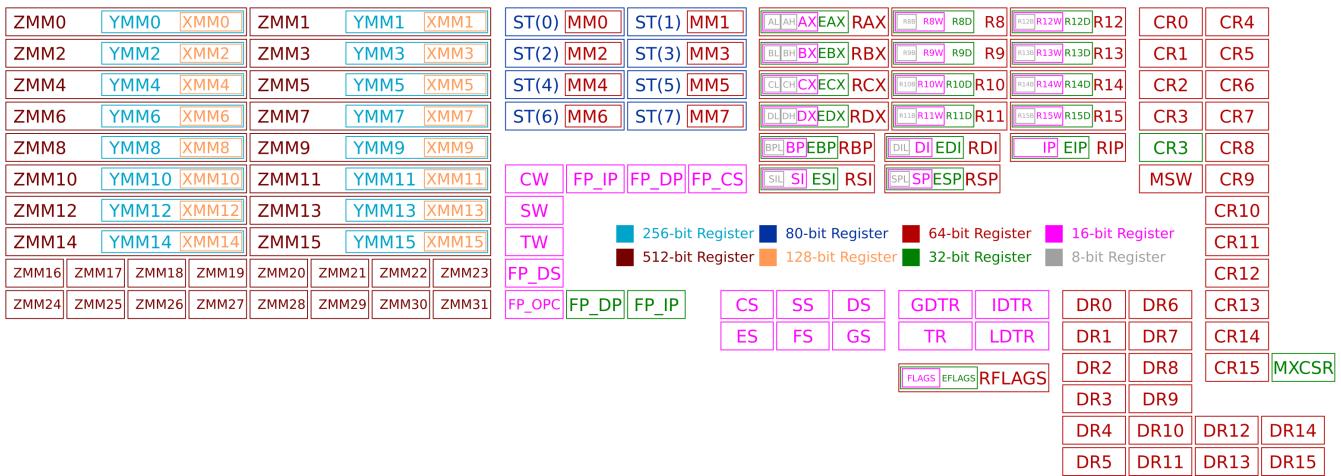
*Logo 10. Jerarquía de Memoria*

- Capacidad vs Latencia
- Concepto de Caching

Registros

*Logo 11. Registros x86*

Registros

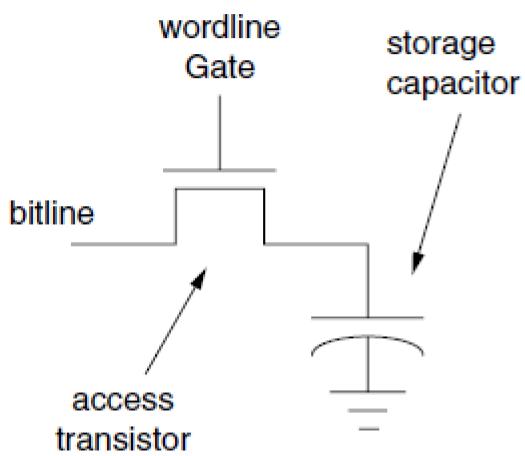


Logo 12. Registros arquitectura amd64

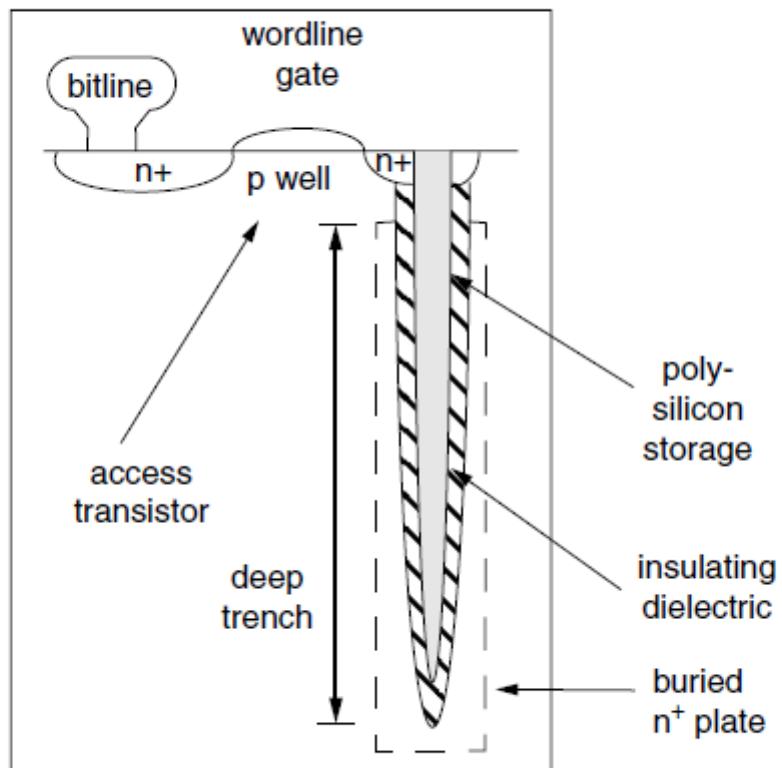
Memoria de Semiconductor

Memory Type	Category	Erasure	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	Nonvolatile
Programmable ROM (PROM)			Electrically	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

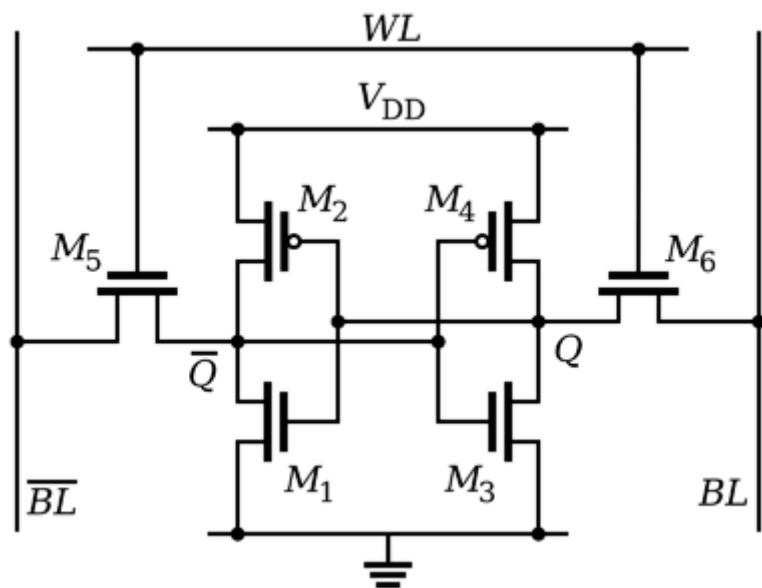
Memoria RAM Dinámica: Celda DRAM



- Celda DRAM: 1 Transistor (14nm) y 1 Condensador (3nm)
- Celda de área reducida y bajo consumo → alto nivel de integración → Memoria Principal
- Fugas de carga del condensador: necesidad de reescritura (Refresco) → Estado Dinámico



Memoria RAM eStática: Celda SRAM

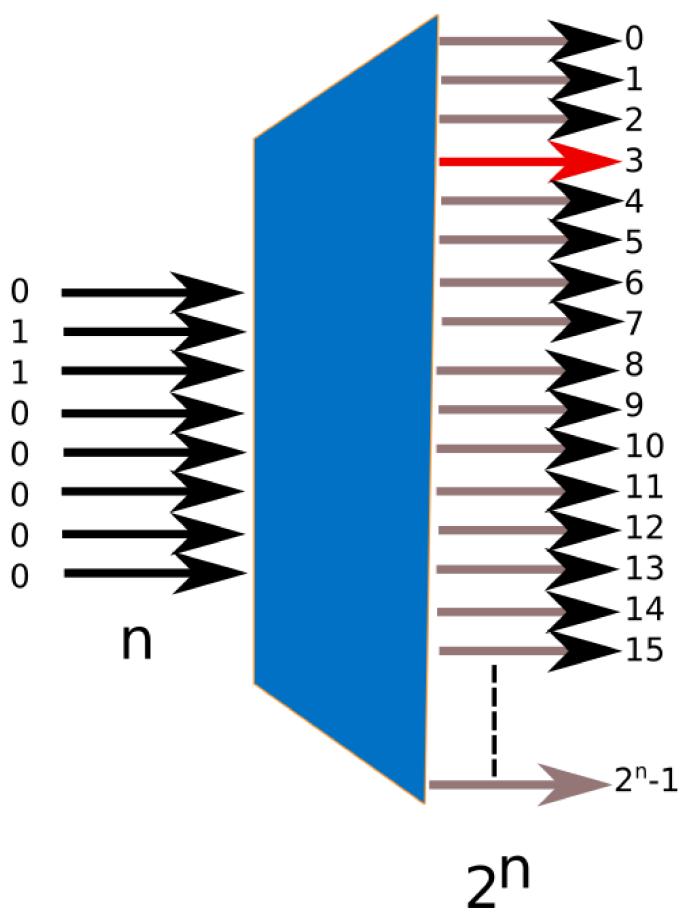


Logo 13. SRAM-cell

- Celda SRAM : 6 Transistores
- Memoria Caché: latencia reducida
- Sin fugas: Estado Estático

Memoria DRAM: Demultiplexor

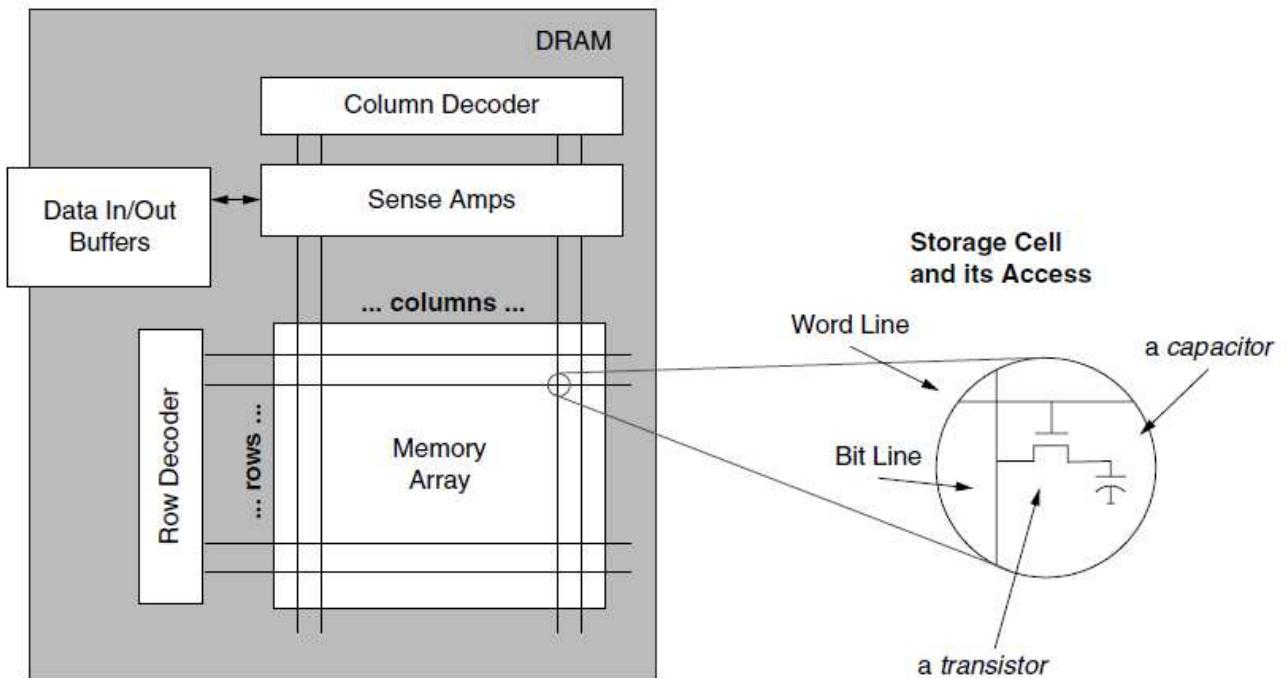
- ¿Cómo se pueden organizar y estructurar miles de millones de celdas?
- Una memoria con un bus de direcciones de 30 hilos qué decodificador necesitaría?
 - Con 30 hilos se pueden direccionar 2^{30} bits
 - Entrada binaria: 30 hilos
 - Salida decimal: $2^{30} = 1.073.741.824$ salidas → IMPOSIBLE DE FABRICAR en un chip



Logo 14. Demultiplexor del bus de Direcciones

Memoria DRAM: Array Bidimensional

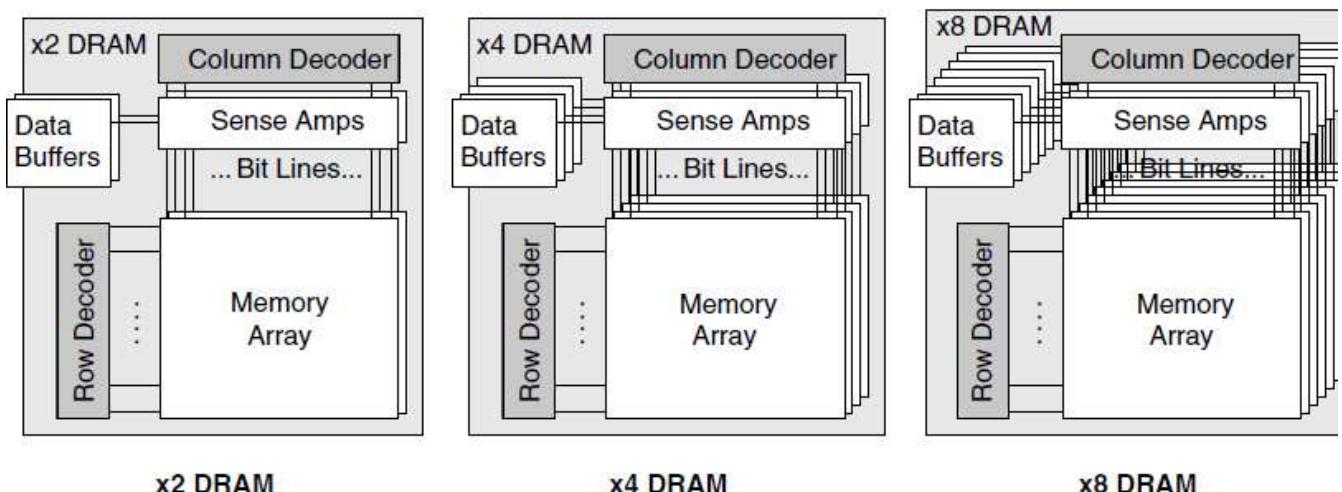
- Solución
 - $2^{30} = 2^{15} * 2^{15}$
 - $2^{15} = 32.768$
 - La solución son dos multiplexores de 15 entradas y 32.768 salidas cada uno.
 - La organización de las celdas es una matriz de dos dimensiones. Filas y Columnas cuya intersección es una celda de memoria.
 - De los 30 hilos del bus de direcciones 15 son la entrada del multiplexor de Filas y las otras 15 del multiplexor de columnas
- Otros Componentes: Amplificador de Salida y Buffer I/O de datos



Logo 15. Array 2D

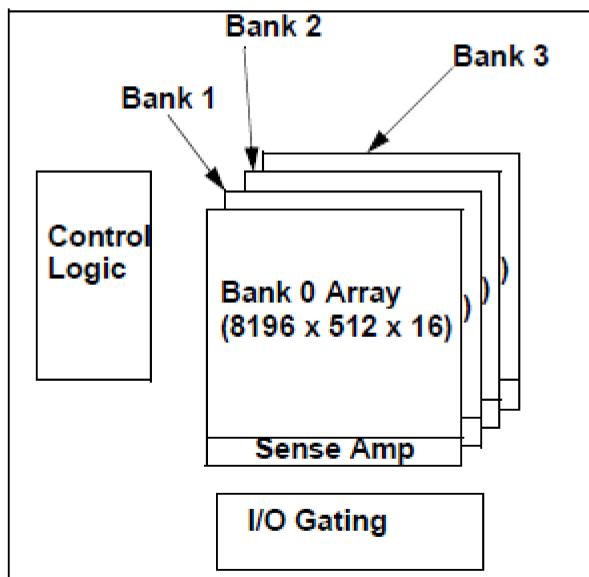
Memoria DRAM: Banco

- Con un Array tenemos datos de un sólo bit
- Mediante el agrupamiento de N arrays tenemos datos de N bits → xN
- Agrupando múltiples arrays organizamos las celdas de memoria en 3 dimensiones
- Un agrupamiento de arrays recibe el nombre de **Bank**



Memoria DRAM: Chip

- Un chip de memoria consiste en el agrupamiento de múltiples Bancos
- Un chip de memoria tiene un **Buffer I/O** para todos los bancos. Almacena el dato a escribir o leer
- El tamaño del Buffer I/O se indica mediante el prefijo xN

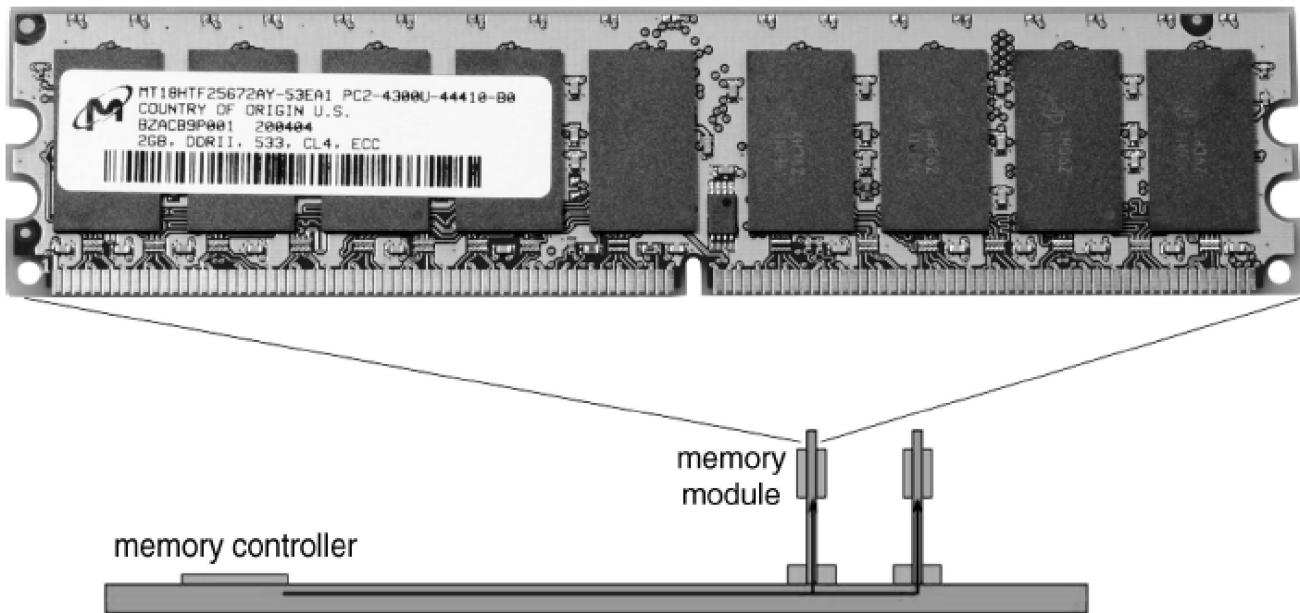


Logo 16. Chip

Memoria DRAM: Ejemplo de un Chipx16

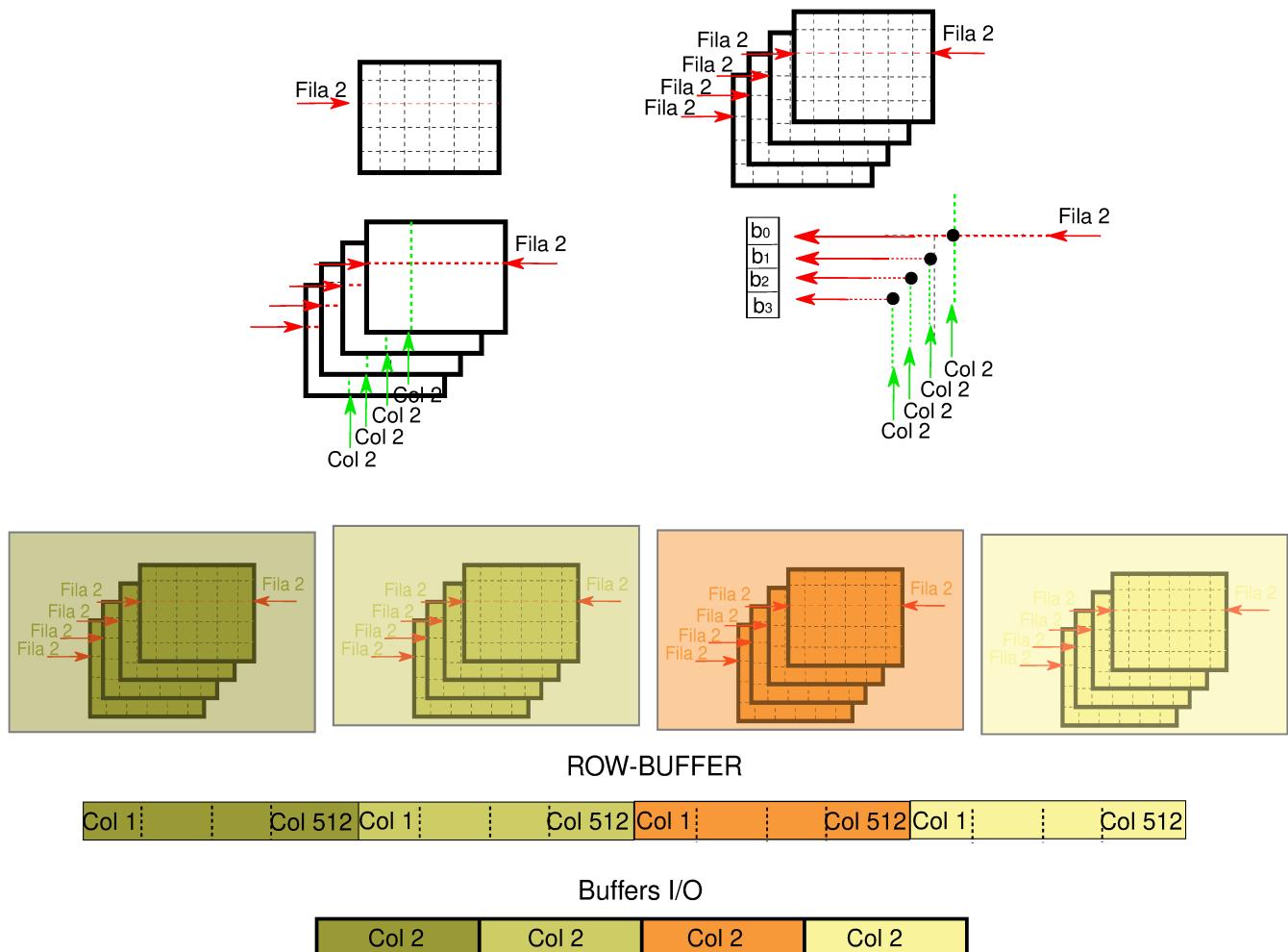
- Control Logic: Circuitería para Direccionamiento, Lectura, Escritura, Refresco, ...
- Un banco \rightarrow 3D: $8196 \times 512 \times 16 \rightarrow$ Filas x Columnas x N°_de_Arrays/Banco $\rightarrow 2^{13} \times 2^9 \times 16$
 - Demultiplexor del direccionamiento de Filas : 13 entradas y 8196 salidas
 - Demultiplexor del direccionamiento de Columnas: 9 entradas y 512 salidas
 - Capacidad de Memoria del Banco: $2^{22} \times 16\text{bits} = 2^{22} \times 2^1\text{Bytes} = 2^{23} \text{ Bytes} = 2^3 \times 2^{20} = 8\text{MB}$
 - Bus de direcciones: $2^{23} \rightarrow 23$ hilos
 - **Espacio de Direcciones** \rightarrow (Fila,Columna) \rightarrow Se seleccionan 2 Bytes de forma NO Lineal.
- **Ejemplo:** Dirección Lineal 0x7A4E6B de un byte
 - Espacio no lineal: (Fila 111-1010-0100-11 , Columna 10-0110-1010) \rightarrow (0x1E93 , 0x26B)
 - Se seleccionan 2 bytes para las direcciones lineales 0x7A4E6A y el 0x7A4E6B ubicados en la posición (0x1E93 , 0x26B) del Banco
- Chip
 - Buffer i/o del chip= 16 bits de datos = 2Bytes
 - Capacidad de Memoria del chip: 4 Bancos x 8MB = 32MB
 - Espacio de direcciones NO lineal (**Banco,Fila,Columna**)
 - La lógica electrónica del Chip debe de realizar la conversión: espacio lineal \rightarrow espacio no lineal

Memoria DRAM: Tarjeta ó Módulo DIMM



- DIMM: Dual In-Line Memory Module : Chips por las dos caras de la tarjeta

Row Buffer

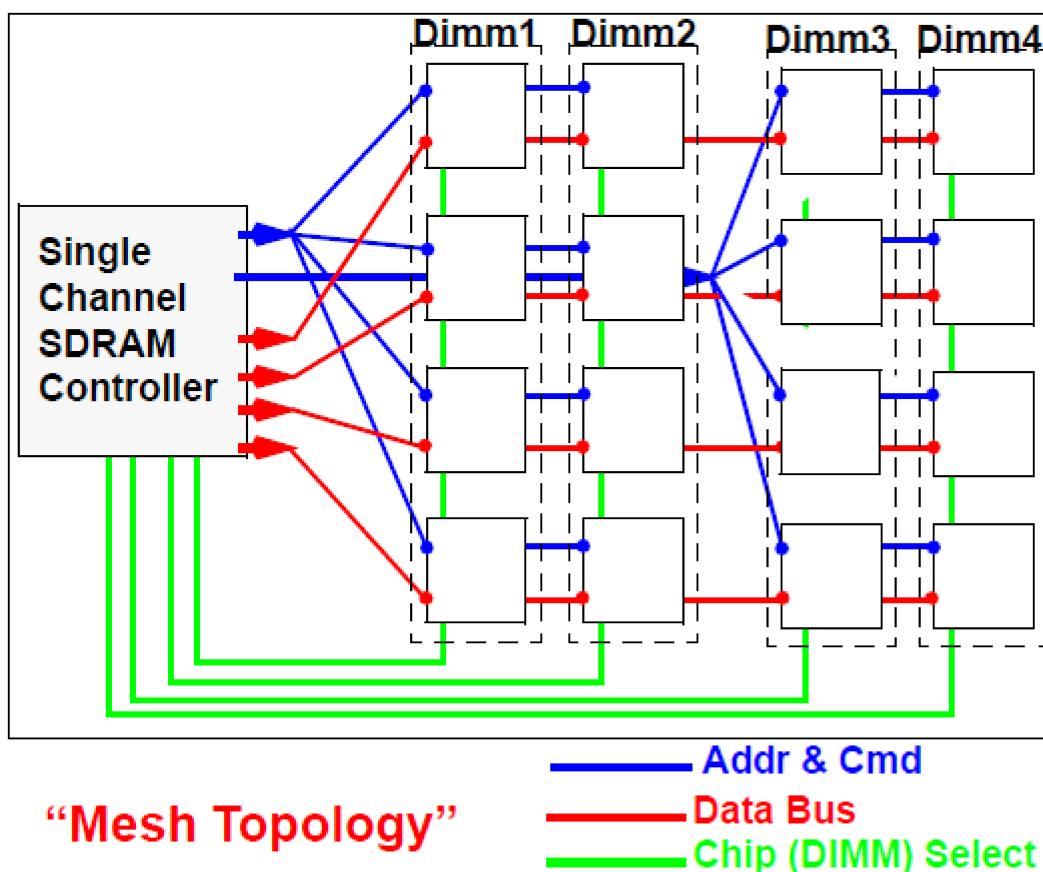


Row Buffer

- Actúa como caché dentro del propio módulo de memoria DRAM
- Capturas sucesivas de operandos → no es necesario volver a leer los condensadores
- Optimización al programar estructuras de datos

Memoria DRAM: Rank

Memory System Organization



Chip Select: Activa los chips para que se conecten a los buses

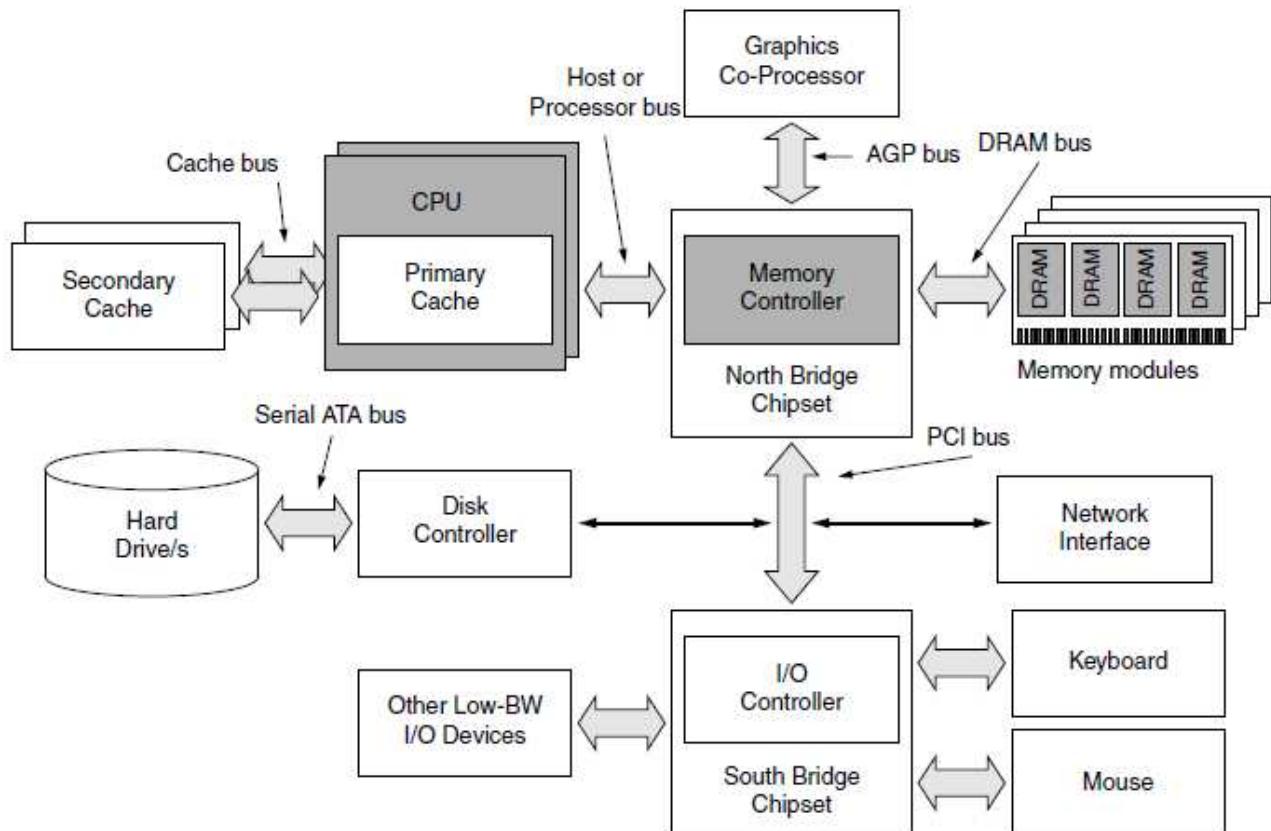
Espacios de Direcccionamiento de Memoria

- Espacio de Direcciones: Formato del modo de direccionamiento
 - Ejemplo no lineal: Prof Cándido Aramburu, Dpto Ingeniería Eléctrica y Comunicaciones, Campus Arrosadía, UPNA, Av/Cataluña, Pamplona 31006, Navarra
 - Ejemplo lineal : DNI 17.987.XXX
- Programador lenguaje ensamblador: espacio de memoria en secciones, etiquetas, ...
- Programa lenguaje binario : espacio de memoria virtual → direccionamiento lógico no lineal → (sección datos , dirección 0x000012AF)
- CPU: espacio de memoria virtual → direccionamiento lógico no lineal → (segmento datos, dirección 0xBC007100)

- Módulo DRAM → espacio (rank, bank, row, column)

Controlador de Memoria

- Unidad Intermedia entre la CPU y los módulos de Memoria DRAM
- Realiza tareas de: Refresh, Sincronismo de la transferencia de datos, Traductor de espacios de direcciones: virtual, lineal, no lineal, etc



Memoria DRAM: Channel

- Un canal de memoria selecciona un grupo de tarjetas DIMM. Cada canal tiene su propio espacio de direccionamiento.
- El Controlador de Memoria puede operar simultáneamente con diferentes canales

Controlador de Memoria DRAM

- Unidad Memory Management Unit (MMU): traduce la dirección virtual lógica en una dirección física lineal
- Traductor físico lineal a físico no lineal (canal, rank, bank, row, column)

Intel® 82946GZ Graphics and Memory Controller Hub

El manual del controlador de memoria de Intel (Intel ® 946 Express Chipset Family año 2006) en el apartado 1.3.2 (System Memory Interface) dice:

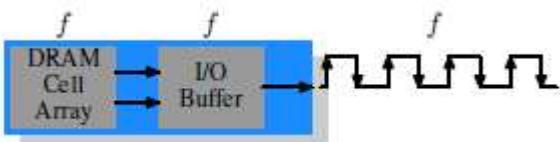
- The (G)MCH integrates a system **memory** DDR2 (Double Data Rate 2 generation) **controller** with two, 64-bit wide interfaces. Capabilities of the system memory interface include:
- Supports 256-Mb, 512-Mb, and 1-Gb technologies for x8 and x16 devices
- Supports four banks for all DDR2 devices up to 512-Mbit density. Supports eight banks for 1-Gbit DDR2 devices

- By using 1 Gb technology in Dual Channel Interleaved Mode, the largest **memory capacity** possible is 4 GB. (16 K rows x 1 K columns x 1 cell/(row x column) x 8 b/cell x 8 banks/device x 8 devices/DIMM-side x 2 DIMM-sides/channel x 2 channels x 1 B/8 b x 1 G/1024 M x 1 M/(KxK)
 $= (2^{14} \text{ rows} \times 2^{10} \text{ columns} \times 1 \text{ cell}/(\text{row} \times \text{column}) \times 2^3 \text{ b/cell} \times 2^3 \text{ banks/device} \times 2^3 \text{ devices/DIMM-side} \times 2^1 \text{ DIMM-sides/channel} \times 2^1 \text{ channels} \times 1 \text{ Byte}/8\text{bits} \times 1 \text{ KB}/1024\text{B} \times 1 \text{ MB}/1024\text{KB} \times 1 \text{ GB}/1024\text{MB} = 2^{(14+10+3+3+3+1+1-3-10-10)} \text{ GB} = 2^{(35-33)} \text{ GB} = 4 \text{ GB}$
- Maximum DRAM address decode space is 4 GB (assuming 36-bit addressing)
- Supports up to 32 simultaneous **open pages** per channel (assuming 4 ranks of 8 bank devices)

Sincronismo de la Memoria DRAM : SDRAM

- La transferencia de datos entre la CPU y los módulos DRAM se realiza de forma síncrona con el reloj propio del bus de memoria.
- El flanco del reloj es el patrón de comienzo y fin de las operaciones
- **DDR (Double Data Rate)**
 - Permite transferir el bit tanto en el flanco de **bajada** como de **subida** del reloj (**doble bombeo**)
 - Arquitectura amd64 → Bus de datos de 64 bits conectado al **Buffer i/o** de la memoria
- Frecuencia del buffer i/o
 - El buffer i/o de la memoria pude ir a frecuencias x2, x4 y x8 respecto de la frecuencia de acceso a la celda.
 - **Supercelda ó Macrocelda:** Ahora una selección (fila,columna) de un array supone no la seleccion de 1 celda sino la de una macrocelda de 2, 4 u 8 CELDAS de cada Banco.
- DDR1: una macrocelda de 2^1 celdas → 2 celdas
 - 1^a Generación: año 2000
 - El buffer i/o del chip tiene 2 registros, cada uno de 64 bits.
- DDR2: una macrocelda de 2^2 celdas → 4 celdas
 - 2^a Generación: año 2006
 - El buffer i/o del chip tiene 4 registros
- DDR3: una macrocelda de 2^3 celdas → 8 celdas
 - 3^a Generación: año 2011
 - El buffer i/o del chip tiene 8 registros
- DDR4: una macrocelda de 2^4 celdas → 16 celdas
 - 4^a Generación: año 2014
 - El buffer i/o del chip tiene 16 registros
- DDR5: una macrocelda de 2^5 celdas → 32 celdas
 - 5^a Generación: año 2020
 - El buffer i/o del chip tiene 32 registros

Sincronismo de la Memoria DRAM : Synchronous DRAM → SDRAM



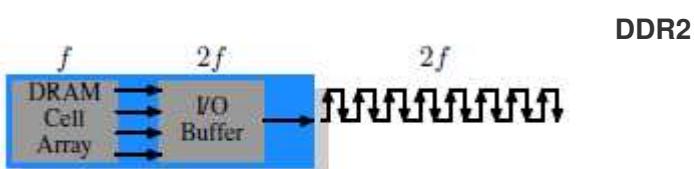
DDR

En cada acceso a la macrocelda se leen 2^1 celdas, es decir, dos datos. Los 2 datos se almacenan en los 2 registros del buffer i/o

En el período del reloj del bus de memoria f_{bm} se realizan 2 transferencias (una en el flanco de bajada y otra en el flanco de subida), a través del bus de datos, de 64 bits (8Bytes) cada una.

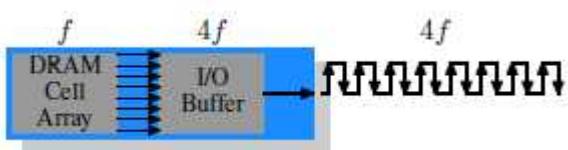
Las 2 transferencias de los 2 registros i/o necesitan por lo tanto 1 período a la frecuencia f_{bm} . Durante el período f_{bm} que dura el vaciado de los 2 registros i/o es necesario el acceso una macrocelda, por lo que la frecuencia de acceso a la macrocelda $f_a = f_{bm}$

La frecuencia efectiva (frecuencia de transferencia de datos de 8Bytes) del bus de memoria f_{ebm} es el doble de la frecuencia f_{bm} por realizarse dos transferencias en un período $\rightarrow f_{ebm} = 2f_{bm} = 2f_a \rightarrow f_{ebm}$ transferencias por segundo $\rightarrow f_{ebm} * 8\text{Bytes}$ por segundo



DDR2

Sincronismo de la Memoria DRAM : Synchronous DRAM → SDRAM



DDR3

En cada acceso a la macrocelda se leen 2^3 celdas, es decir, 8 datos. Los 8 datos se almacenan en los 8 registros del buffer i/o

En el período del reloj del bus de memoria f_{bm} se realizan 2 transferencias (una en el flanco de bajada y otra en el flanco de subida), a través del bus de datos, de 64 bits (8Bytes)cada una.

Las 8 transferencias de los 8 registros i/o necesitan por lo tanto 4 períodos a la frecuencia f_{bm} .

Durante los 4 períodos f_{bm} que dura el vaciado de los 8 registros i/o es necesario el acceso una macrocelda, por lo que la frecuencia $f_{bm} = 4f_a$

La frecuencia efectiva del bus de memoria f_{ebm} es el el doble de la frecuencia f_{bm} por realizarse dos transferencias en un período $\rightarrow f_{ebm} = 2f_{bm} = 8f_a \rightarrow f_{ebm}$ transferencias por segundo $\rightarrow f_{ebm} * 8\text{Bytes}$ por segundo

Sincronismo de la Memoria DRAM: Nominación de los módulos de memoria

- PC2-6400 (DDR2-800) 5-5-5-16
 - DDR 2^a generación → Macroceldas de 4 celdas → Buffer i/o de 4 registros
 - Ancho de banda BW = 6400 MB/s
 - $f_{ebm} = 800\text{MHz} \rightarrow BW=8\text{Bytes/transferencia} * 800 * 10^6 \text{transferencias/seg} = 6400 \text{MB/s}$
 - $f_{bm}=f_{ebm}/2=400\text{MHz} \rightarrow f_a=f_{ebm}/4=200\text{MHz}$
 - 5-5-5-16 : diferentes tiempos de sincronismo (tCL-tRCD-tRP-tRAS) del bus de control expresados en número de ciclos de reloj ($1/f_{bm} = 2.5\text{ns}$) $\rightarrow tRAS=16 * 2.5\text{ns}=40\text{ns} \rightarrow$ da una idea del orden del tiempo de latencia o tiempo de acceso a la memoria.

- Si a la CPU le llegan datos a la frecuencia de 800MHz quiere decir que las fases de captura de instrucciones y operandos duran como mínimo 1/800 microsegundos → 1.25 nanosegundos
- PC3-6400 (DDR3-800) 5-5-5-10
 - DDR 3^a generación → Macroceldas de 8 celdas → Buffer i/o de 8 registros
 - Ancho de banda BW = 6400 MB/s
 - $f_{ebm} = 800\text{MHz} \rightarrow BW = 8\text{Bytes/transferencia} * 800 * 10^6 \text{transferencias/seg} = 6400 \text{MB/s}$
 - $f_{bm} = f_{ebm}/2 = 400\text{MHz} \rightarrow f_a = f_{ebm}/8 = 100\text{MHz}$
 - 5-5-5-10 : diferentes tiempos de sincronismo (tCL-tRCD-tRP-tRAS) del bus de control expresados en número de ciclos de reloj ($1/f_{bm} = 2.5\text{ns}$) → tRAS=10*2.5ns=25ns

Memoria Cache

Memoria Cache : Arquitectura

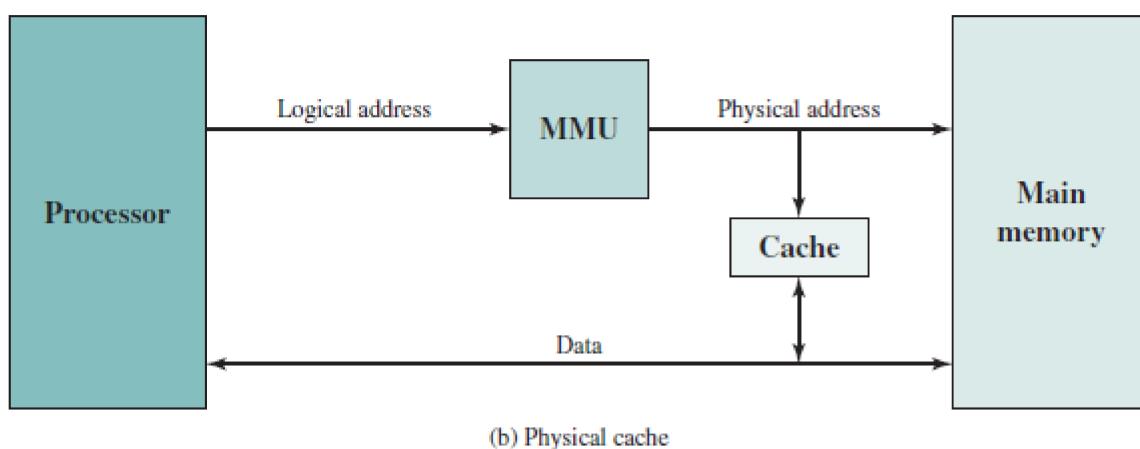
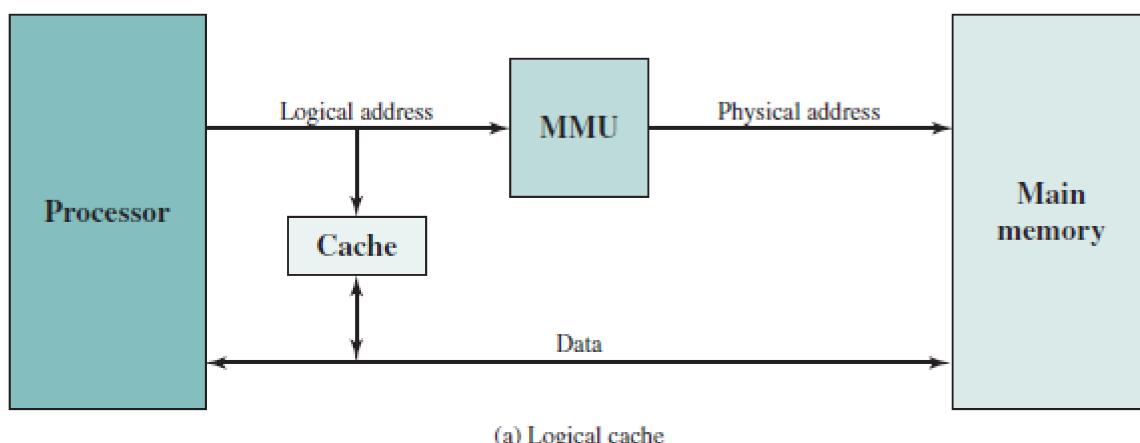


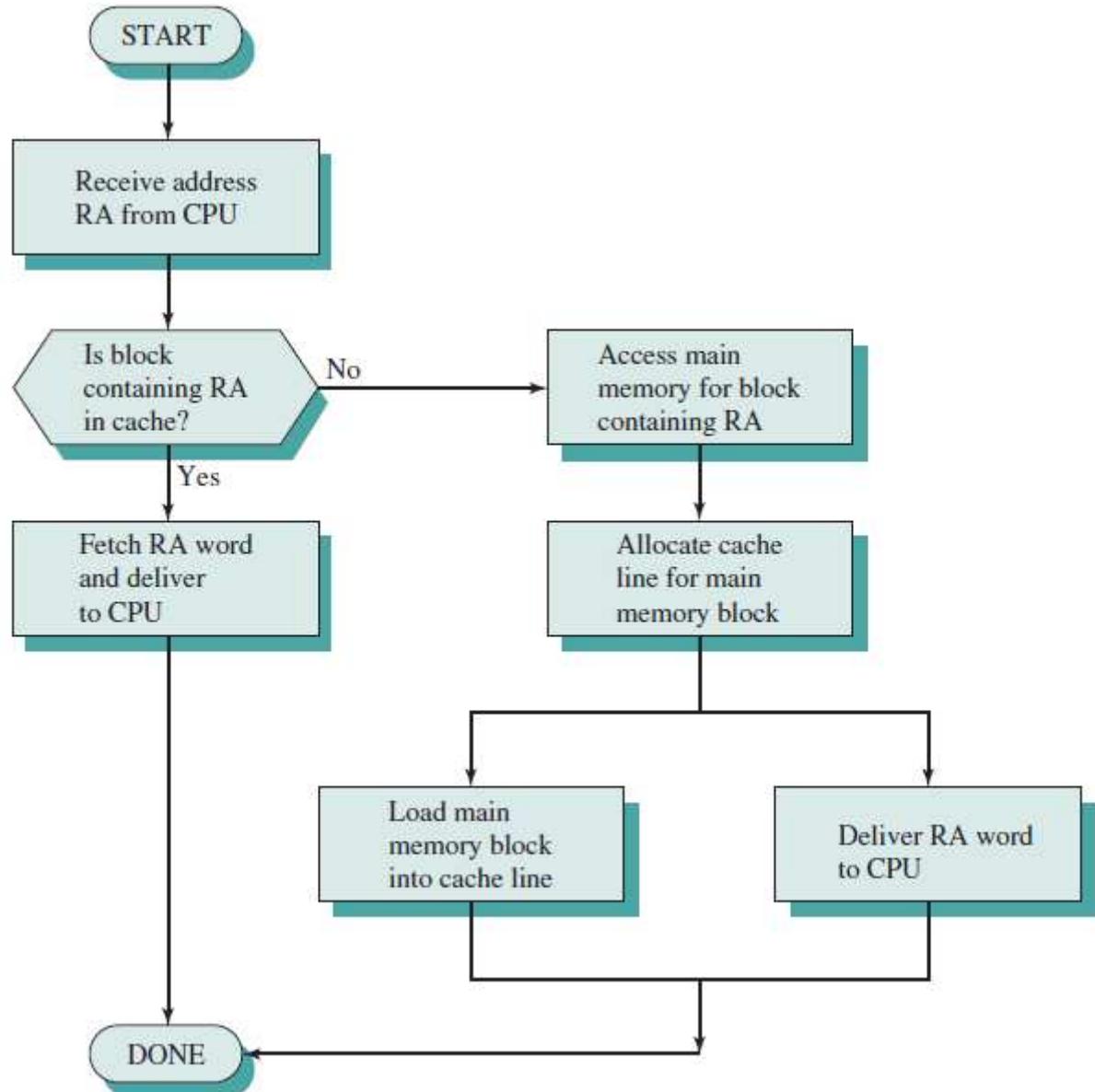
Figure 4.7 Logical and Physical Caches

Memoria Cache : Introducción

- Función:
 - Reducir la latencia de las fases de captura del ciclo de instrucción
- Implementación:
 - Memoria estática → la latencia caché es un 10% la latencia de la memoria DRAM

- Unidades:
 - MMU: Memory Management Unit → traducción de direcciones virtuales (lógicas no lineales) a direcciones físicas (líneales)
 - Memoria dinámica SDRAM → memoria principal
 - Memoria estática SRAM → memoria caché
 - La interfaz entre la CPU y la Memoria SDRAM es el controlador de memoria SDRAM
 - La interfaz entre la CPU y la Memoria Caché es el controlador de memoria caché

Memoria Cache: Read Operation



Memoria Caché: Organización

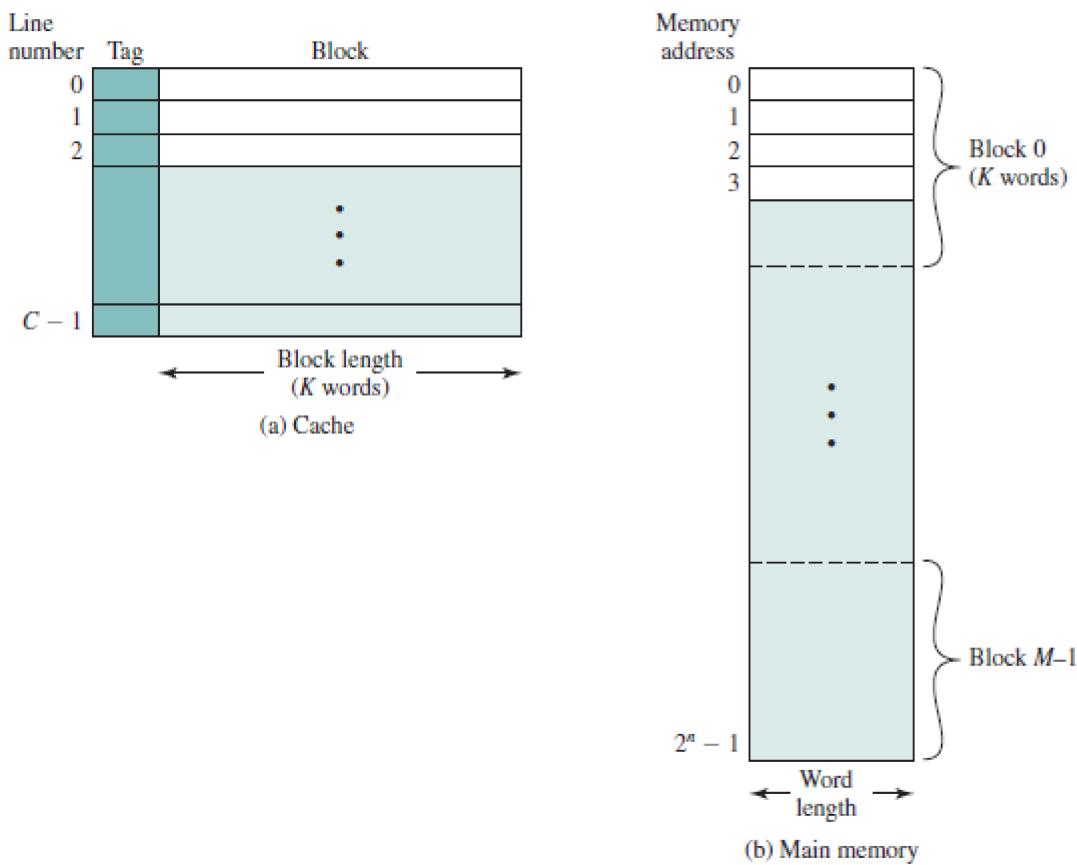


Figure 4.4 Cache/Main Memory Structure

Memoria Caché: bloques, líneas, palabras

- La memoria principal:
 - se organiza en bloques y direcciones de bloque: cada bloque se divide en palabras y cada palabra en bytes
- la memoria caché:
 - se organiza en líneas y direcciones de la línea
 - cada línea de la memoria caché se corresponde con un bloque de la memoria principal según la función de correspondencia del controlador de la caché
 - el bloque de la línea por lo tanto se organiza en palabras y las palabras en bytes
 - cada línea tiene dos campos: la marga TAG y el bloque de la memoria principal correspondiente
 - el campo TAG de una línea informa sobre qué bloque principal corresponde a dicha línea caché

Funciones de correspondencia: directa, totalmente asociativa, asociativa por conjuntos

- Función directa
 - A cada bloque principal le corresponde una **dirección fija** de una línea
- Función totalmente asociativa
 - Cada bloque puede copiarse en **cualquier** dirección de línea
- Función asociativa por conjuntos
 - Cada bloque principal puede copiarse de manera libre dentro un pequeño conjunto de líneas

Ejemplo: Función de correspondencia directa

- Enunciado:
 - m : capacidad de la memoria caché : 64KB
 - Word size : 1 byte
 - K: Palabras / bloque : 4
 - M : capacidad de la memoria principal : 16 MB
 - ¿Correspondencia de la dirección MP 0xA912AB?
- Caché
 - Dirección de byte de la M. caché: $64KB = 2^{16} \rightarrow 16\text{bits}$
 - Dirección de palabra de la M. caché: $64Kword = 2^{16} \rightarrow 16\text{bits}$
 - Dirección línea caché: $4 \text{ palabras/línea } 64Kword/4 = 2^{14} \rightarrow 14\text{bits}$
- MP
 - Formato dirección MP : Bloque-Palabra
 - Dirección MP: $16MB = 2^{24} \rightarrow 24\text{bits}$
 - Dirección Bloque MP: 1 línea \leftrightarrow 1 bloque $\rightarrow 16\text{bits}$
 - Dirección palabra : 4 palabras $\rightarrow 2^2 \rightarrow 2\text{bits}$
- Direccionamiento bloque-palabra
 - los primeros 22 bits son el bloque y los 2 últimos bits la palabra
 - hay 2^{22} bloques que se pueden corresponder con una de las 2^{14} líneas
 - de los 22 bits de bloque los últimos 14 son la dirección de línea y los 8 primeros son el TAG
- Formato
 - Bloque/Word $\rightarrow 22/2$
 - Tag/Linea/Word $\rightarrow 8/14/2$

Ejemplo: Función de correspondencia directa

- Función de correspondencia directa:
 - el bloque 0 se corresponde con la línea de dirección 0 y así sucesivamente
 - el bloque 1 se corresponde con la línea de dirección 1
 - el bloque $(2^{14}-1)$ se corresponde con la línea de dirección $(2^{14}-1)$
 - el bloque 2^{14} se corresponde con la línea de dirección 0
 - el bloque "j" se corresponde con la línea de dirección $i = j \text{ MOD}[m]$ donde m es la capacidad de líneas de la M. caché.

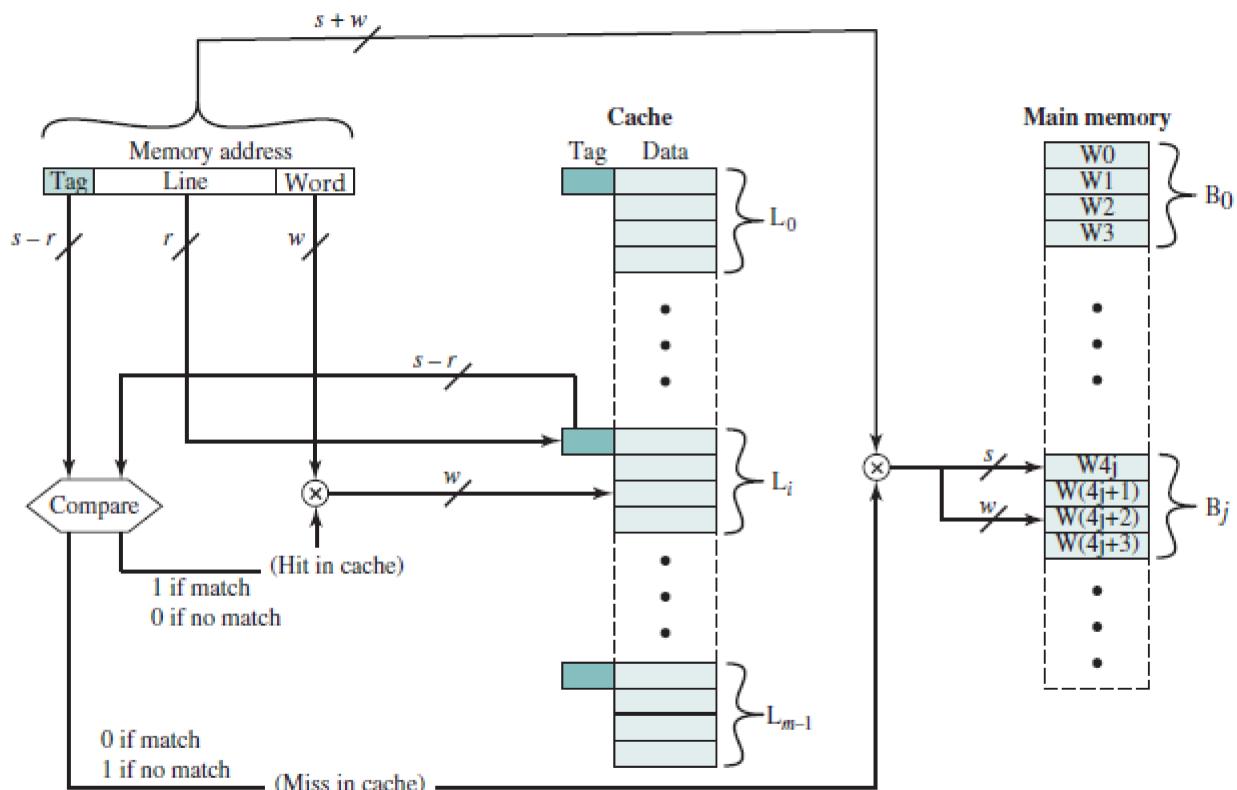


Figure 4.9 Direct-Mapping Cache Organization

Ejemplo: Función de correspondencia directa

- Direcciones de los bloques
 - el bloque "j" se corresponde con la dirección $j * 4$
 - multiplicar por 2^2 equivale a un desplazamiento de 2 bits hacia la izda
 - dividir por 2^2 equivale a un desplazamiento de 2 bits hacia la derecha
- Dirección MP 0xA912AB → 1010 1001 0001 0010 1010 1011
 - Bloque/palabra → 22/2
 - Dirección de bloque 1010 1001 0001 0010 1010 10 → 10 1010 0100 0100 1010 1010 → 0x2A44AA
 - Dirección de palabra 11 → 3
 - Tag/línea/Word → 8/14/2
 - Método 1º: Dirección de línea 0001 0010 1010 10 → 00 0100 1010 1010 → 0x04AA
 - Método 2º: 0x2A44AA MOD $[2^{14}]$ → Resto de 0x2A44AA / 2^{14}
 - $0x2A44AA / 2^{14} = 0xA9.12A8 \rightarrow$ Resto=0x0.12A8* $2^{14}=14$ bits MSb de 0x12A8 → 0001 0010 1010 10
 - Divi=Coc*Disor+Resto → Divi/Coc=Disor+Resto/Coc → Resto=Fracción*Cociente

Ejemplo: Función de correspondencia Asociativa Total

- No hay función de correspondencia ya que a los bloques principales se le puede cualquier línea.
- Se le correspondería una línea que no estuviera ocupada por un bloque
- Si están todas ocupadas es necesaria una "política de reemplazo": Por ejemplo la línea que más tiempo lleva sin ser accedida por la CPU conocida como política "least-recently used (LRU)"

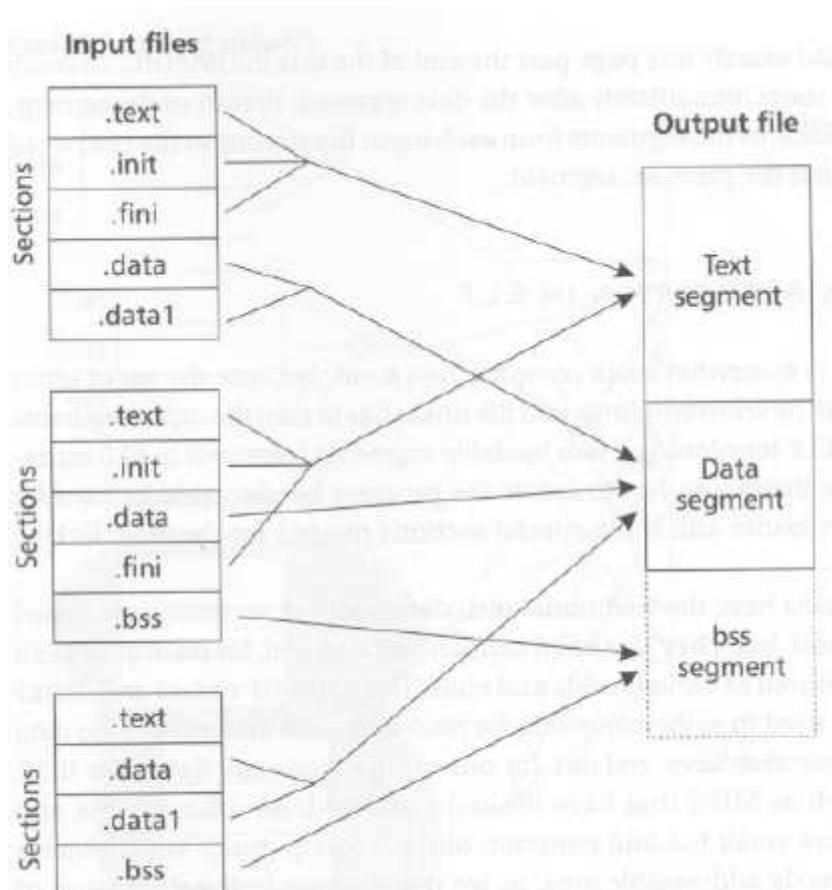
Política de Escritura en la Caché

- Si la CPU escribe un resultado en una línea de la caché, su bloque principal correspondiente no será idéntico por lo que será necesario copiar la línea de la caché en el bloque principal correspondiente. Es necesaria una política de reescritura para indicar cuál es el momento de realizar dicha copia, si justo después de haber modificado la línea de la caché o más tarde.

Memoria Virtual: Introducción

- ¿Qué es? ¿dónde está?
- Programador
 - Etiquetas, punteros, etc
 - Compilador, Linker → módulo binario ejecutable
- Disco

Memoria Virtual lógica: Secciones y Segmentos

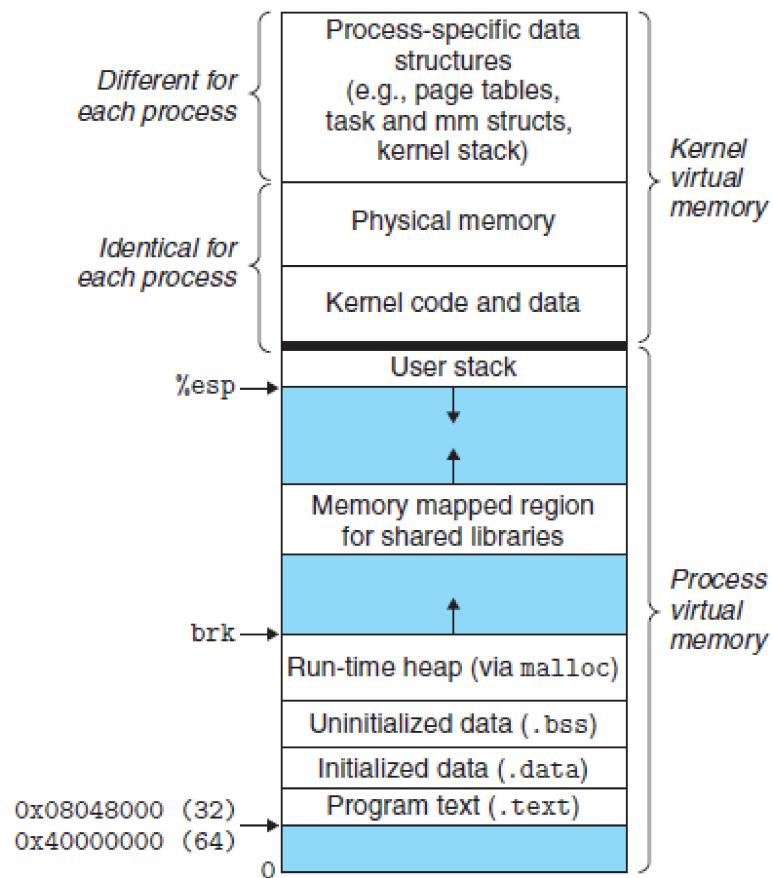


La memoria virtual de un programa binario está organizada en segmentos de datos, instrucciones, pila, heap, etc. La memoria virtual de los módulos binarios antes de ser linkados están organizados en secciones de datos, instrucciones, pila, heap, etc. El linker resuelve los valores de direcciones de las etiquetas de cada sección y los entrelaza. Posteriormente el linker agrupa las diferentes secciones en segmentos.

FIGURE 4.9 • ELF linking.

Memoria Virtual lógica: Mapa de Memoria en un entorno Linux

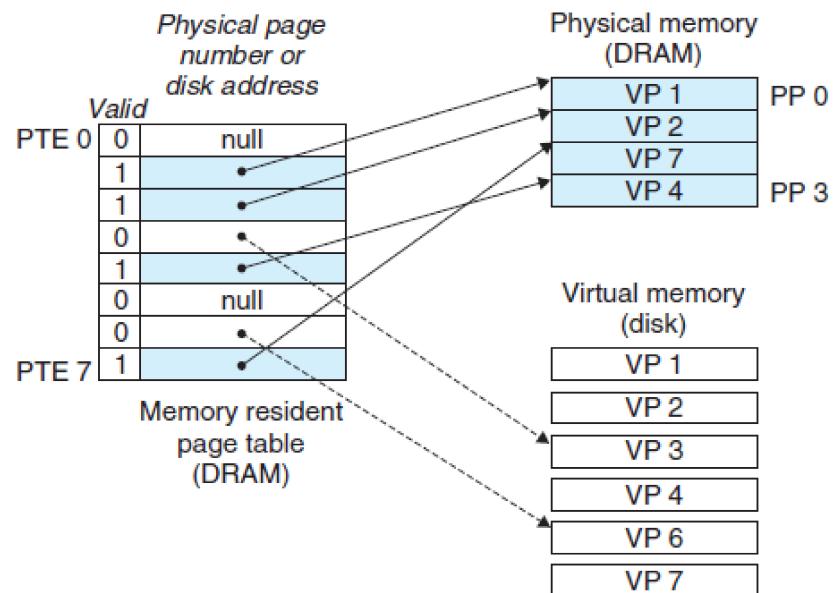
Figure 9.26
The virtual memory of a Linux process.



Memoria Virtual lógica: Paginación

El espacio de direcciones virtual es un espacio abstracto que se partitiona de forma lógica en páginas y la unidad MMU traduce las direcciones de página virtual en direcciones físicas de la memoria DRAM

Figure 9.4
Page table.



Ejercicios: Libro de texto "Computer Organizations 7^aed, Stalling, Williams

Tema 1. Arquitectura de Von Neumann: Capítulo 10.2: Ejercicios 13,14

Tema 7. CPU: Capítulo 10.13 (Processor Structure and Function): Ejercicios 3,7,8

Capítulo 10.14 (RISC): Ejercicio 1

Tema 9. Memoria: DRAM (Ejemplos en los apuntes de teoría)

Cache : Capítulo 10.9: Ejercicios 1,2,3,4,5

Tema 8. Entrada/Salida: Capítulo 10.2 (Arquitectura Von Neuman): Ejercicio 9

Capítulo 10.11 (Entrada/Salida): Ejercicios 1,3

Ejercicios

** Arquitectura de Von Neumann: Capítulo 10.2

13- The Intel 8088 microprocessor has a read bus timing similar to that of Figure 3.19, but requires four processor clock cycles. The valid data is on the bus for an amount of time that extends into the fourth processor clock cycle. Assume a processor clock rate of 8 MHz.

14- The Intel 8086 is a 16-bit processor similar in many ways to the 8-bit 8088. The 8086 uses a 16-bit bus that can transfer 2 bytes at a time, provided that the lower-order byte has an even address. However, the 8086 allows both even- and odd-aligned word operands. If an odd-aligned word is referenced, two memory cycles, each consisting of four bus cycles, are required to transfer the word. Consider an instruction on the 8086 that involves two 16-bit operands. How long does it take to fetch the operands? Give the range of possible answers. Assume a clocking rate of 4 MHz and no wait states

** Processor Structure and Function: Capítulo 10.13

3- A microprocessor provides an instruction capable of moving a string of bytes from one area of memory to another. The fetching and initial decoding of the instruction takes 10 clock cycles. Thereafter, it takes 15 clock cycles to transfer each byte. The microprocessor is clocked at a rate of 10 GHz.

a)Determine the length of the instruction cycle for the case of a string of 64 bytes.

b)What is the worst-case delay for acknowledging an interrupt if the instruction is noninterruptible?

c)Repeat part (b) assuming the instruction can be interrupted at the beginning of each byte transfer

7- A pipelined processor has a clock rate of 2.5 GHz and executes a program with 1.5 million instructions. The pipeline has five stages, and instructions are issued at a rate of one per clock cycle. Ignore penalties due to branch instructions and out-of-sequence executions.

a)What is the speedup of this processor for this program compared to a nonpipelined processor, making the same assumptions used in Section 12.4?

b)What is throughput (in MIPS) of the pipelined processor?

8- Consider an instruction sequence of length n that is streaming through the

instruction pipeline. Let p be the probability of encountering a conditional or unconditional branch instruction, and let β be the probability that execution of a branch instruction I causes a jump to a nonconsecutive address. Assume that each such jump requires the pipeline to be cleared, destroying all ongoing instruction processing, when I emerges from the last stage. Revise Equations (12.1) and (12.2) to take these probabilities into account.

** RISC: Capítulo 10.14

1- We wish to determine the execution time for a given program using the various pipelining schemes discussed in Section 13.5. Let N = number of executed instructions, J = number of jump instructions, D = number of memory accesses. For the simple sequential scheme (Figure 13.6a) for a RISC architecture, the execution time is $2N+D$ stages. Derive formulas for two-stage, three-stage, and four-stage pipelining.

** Cache : Capítulo 10.9

1- Calcular para una controladora de Caché con Asociación de Mapeo Directo, qué línea de la memoria caché (tag/línea/word) está asociada a la dirección 0x02ABCD de la memoria principal.

2- Repetir el ejercicio anterior mediante multiplicaciones y divisiones binarias respondiendo a los siguientes apartados:

3- The system has a Cache memory of 64KB and Main Memory of 16MB with a byte word size and four word block size. For a cache controller with direct mapping correspondence function search the main memory block addresses correspondences to cache memory 0x0CE7 number line .

4- A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 words each. Show the format of main memory addresses. Calculate the 0x7FAFA memory address show in hexadecimal code the cache format. How know the cache controller if the 0x7FAFA is in cache memory.

5- For the hexadecimal main memory addresses 111111, 666666, BBBBBB, show the following information, in hexadecimal format:

**Arquitectura Von Neumann: Capítulo 10.2:

9

** Entrada Salida :Capítulo 10.11:

1- On a typical microprocessor, a distinct I/O address is used to refer to the I/O data registers and a distinct address for the control and status registers in an I/O controller for a given device. Such registers are referred to as ports. In the Intel 8088, two I/O instruction formats are used. In one format, the 8-bit opcode specifies an I/O operation; this is followed by an 8-bit port address. Other I/O opcodes imply that the port address is in the 16-bit DX register. How many ports can the 8088 address in each I/O addressing mode? .

3- A system is based on an 8-bit microprocessor and has two I/O devices. The I/O controllers for this system use separate control and status registers. Both devices handle data on a 1-byte-at-a-time basis. The first device has two status lines and three control lines. The second device has three status lines and four control lines.

How many 8-bit I/O control module registers do we need for status reading and

control of each device?

What is the total number of needed control module registers given that the first device is an output-only device?

How many distinct addresses are needed to control the two devices?

Consider a hypothetical microprocessor generating a 16-bit address (for example, assume that the program counter and the address registers are 16 bits wide) and having a 16-bit data bus.

What is the maximum memory address space that the processor can access directly if it is connected to a "16-bit memory"?

What is the maximum memory address space that the processor can access directly if it is connected to an "8-bit memory"?

What architectural features will allow this microprocessor to access a separate "I/O space"? Many 8-bit I/O ports can the microprocessor support? How many 16-bit I/O ports? Explain.