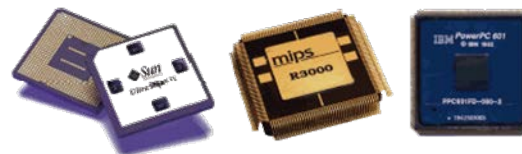


Tema 1: Introducción

Fundamentos de Ordenadores y Sistemas Operativos

Mario Martínez Zarzuela
marmar@tel.uva.es



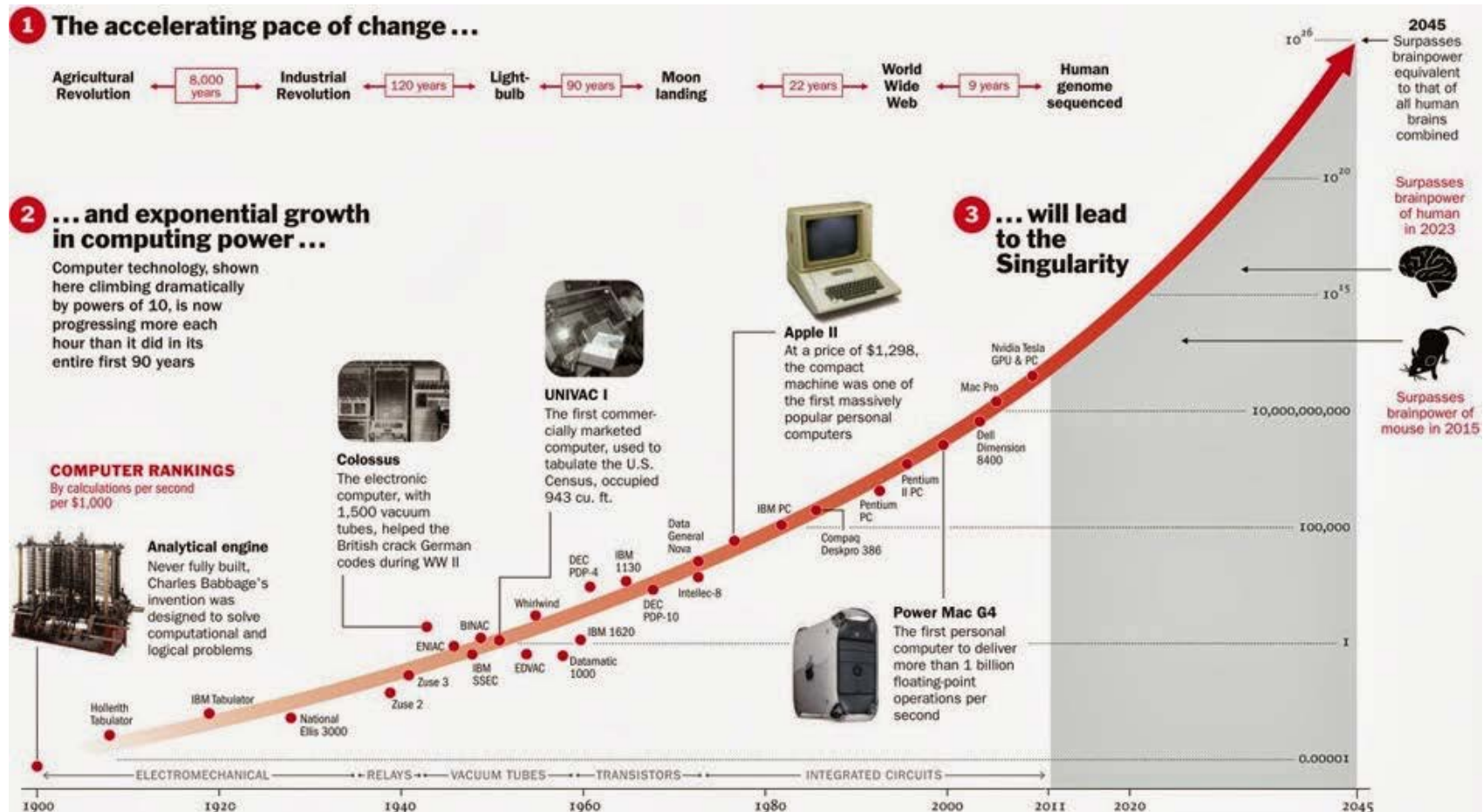
Preguntas a responder

1. ¿Cuáles son las diferencias fundamentales entre los primeros ordenadores y los actuales?
2. ¿Cómo se almacena la información en los ordenadores?
3. ¿Cómo se genera el código máquina interpretable por el microprocesador?

Ley de los rendimientos acelerados



- Ray Kurzweil



Contenidos

- Evolución hasta arquitecturas de microprocesador
- Generación de código máquina
- Practiquemos código binario y hexadecimal
- Conclusiones y consulta adicional

Máquinas de dos estados

Tubos de vacío → Transistores → Circuitos integrados



Vacuum tubes: slow, expensive, fragile

Transistors: much simpler, much smaller, much cheaper, more reliable, no warm up, much faster.



Integrated circuits: miniaturization added to all the existing benefits, enabled unthought-of possibilities

Laboratorio / No Presencial

ENIAC	
Fecha de fabricación:	
Número de tubos de vacío:	
Número de sumas:	
Número de multiplicaciones:	
Número de divisiones:	
Números decimales en memoria:	

- Visualiza el [vídeo sobre el ENIAC](#) y completa la tabla
- <https://youtu.be/32o4qcYbWMA>



Primeros ordenadores

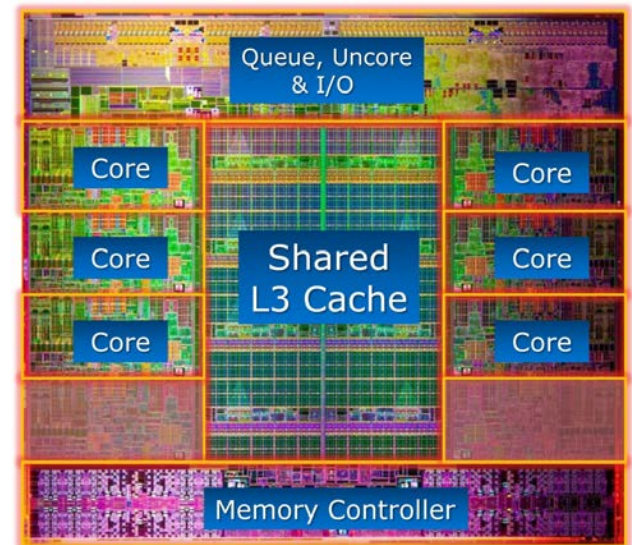
ENIAC	
Fecha de fabricación:	
Número de tubos de vacío:	
Número de sumas:	
Número de multiplicaciones:	
Número de divisiones:	
Números decimales en memoria:	

- Visualiza el [vídeo sobre el ENIAC](#) y completa la tabla



Microprocesador

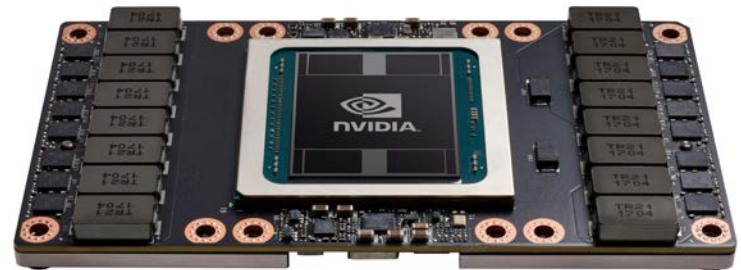
- Circuito electrónico integrado en cuyo interior se alojan uno o más núcleos de procesamiento
- Fabricado con transistores sobre una oblea de Silicio
- Son microprocesadores:
 - CPU: Central Processing Unit
 - GPU: Graphics Processing Unit



Laboratorio / No Presencial

PROCESADOR GPU VOLTA	
Fecha de fabricación:	
Número de transistores:	
Número de núcleos:	
Número de operaciones 32 bits:	
Número de operaciones 64 bits:	
Tecnología de fabricación (nm):	

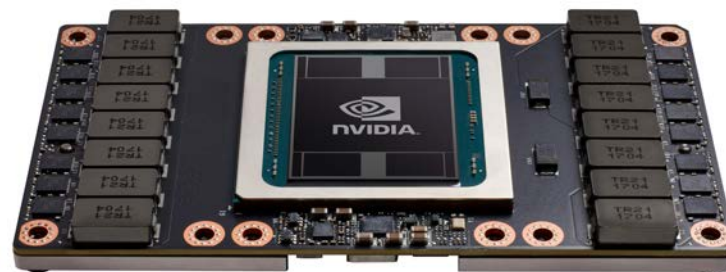
- Visualiza el [vídeo sobre la GPU Volta](#) y completa la tabla
- <https://youtu.be/3aAEKRDhrj8>



Últimos procesadores

PROCESADOR GPU VOLTA	
Fecha de fabricación:	
Número de transistores:	
Número de núcleos:	
Número de operaciones 32 bits:	
Número de operaciones 64 bits:	
Tecnología de fabricación (nm):	

- Visualiza el [vídeo sobre la GPU Volta](#) y completa la tabla



Ley de Moore



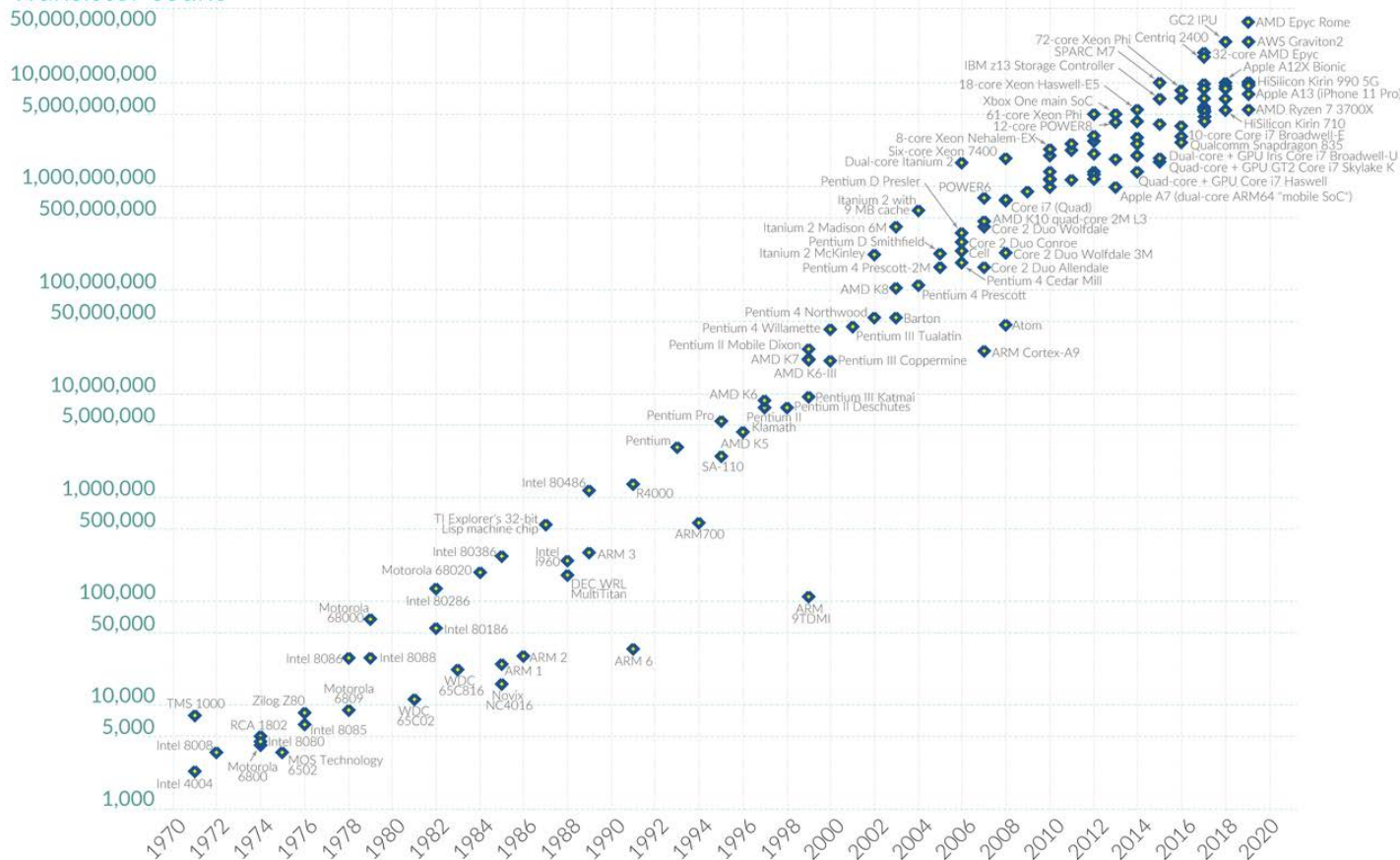
- El número de transistores en un circuito integrado se duplica cada 24 meses

Moore's Law: The number of transistors on microchips doubles every two years

Our World
in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

Transistor count

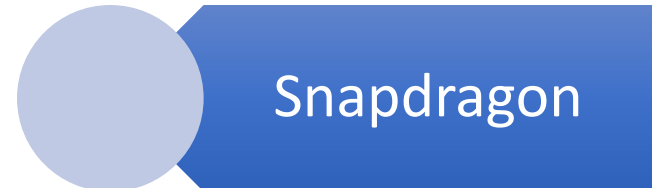
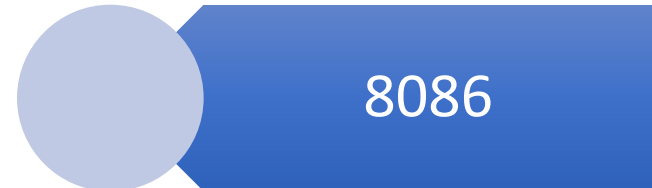
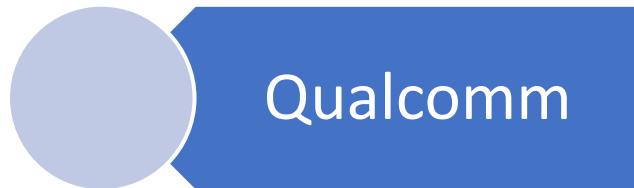


Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)

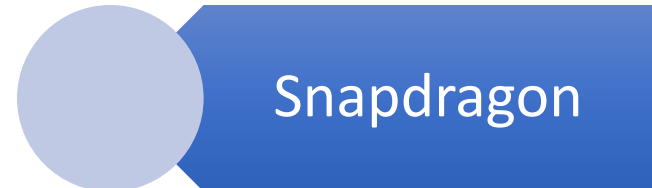
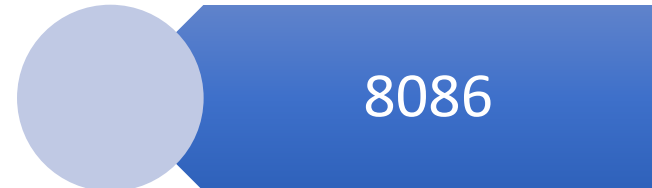
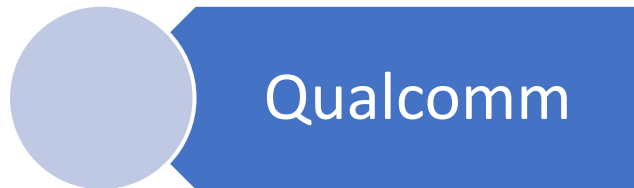
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Licensed under CC-BY by the authors Hannah Ritchie and Max Roser.

¿Cuáles son arquitecturas de microprocesador?



¿Cuáles son arquitecturas de microprocesador?



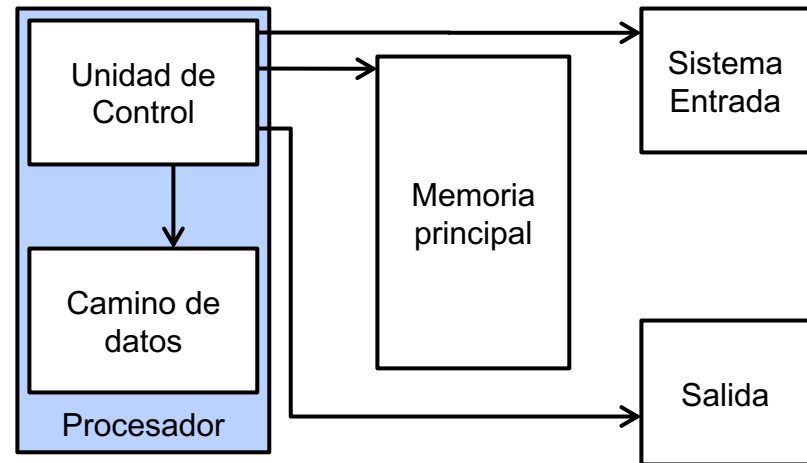
¿En qué se basan las arquitecturas actuales?

- Partes fundamentales:

- Procesador
- Memoria principal
- Sistemas de E/S

- Dentro del procesador:

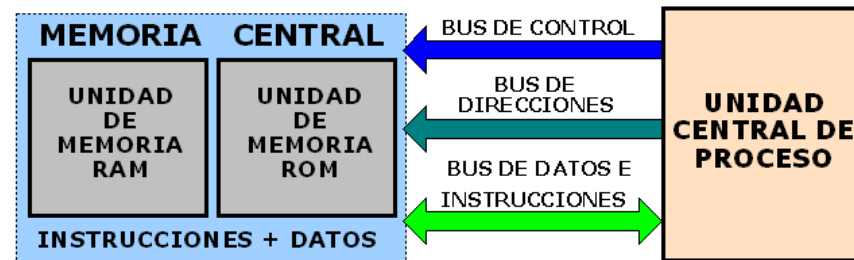
- Camino de datos
- Unidad de Control
- ... y:
 - Registros
 - Memoria caché



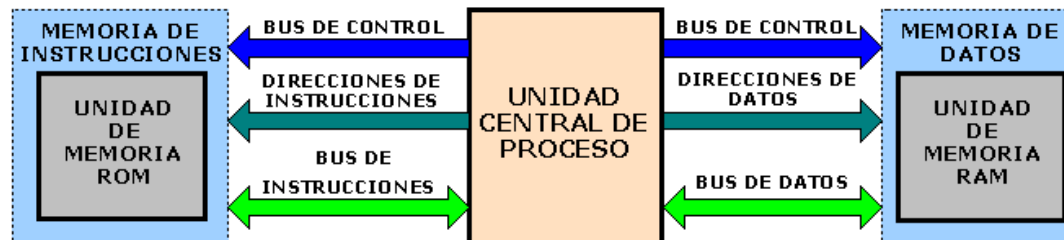
¿En qué se basan las arquitecturas actuales?

- Evolución de arquitecturas Von Neumann y Harvard
 - Programa y datos se encuentran en memoria(s)

ARQUITECTURA VON NEUMANN

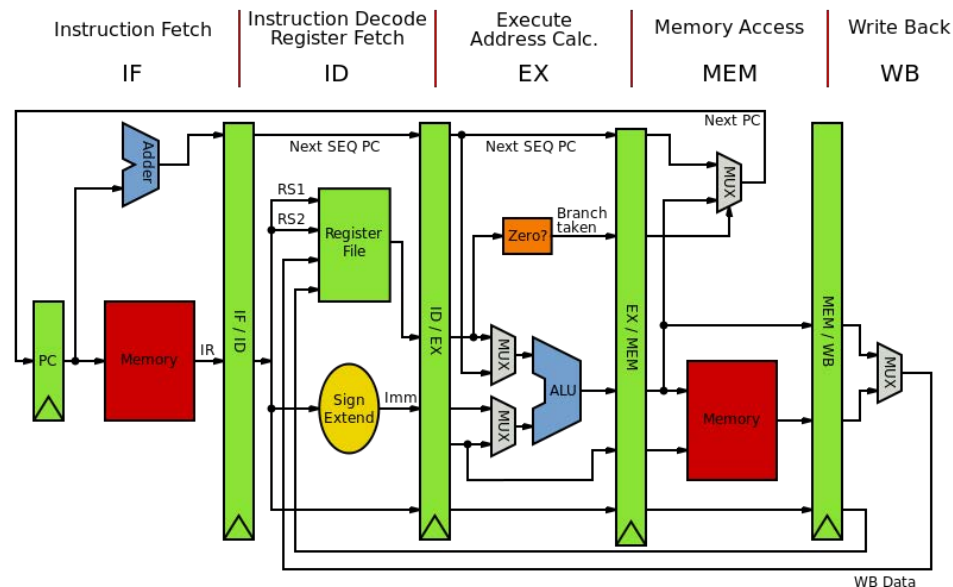


ARQUITECTURA HARVARD



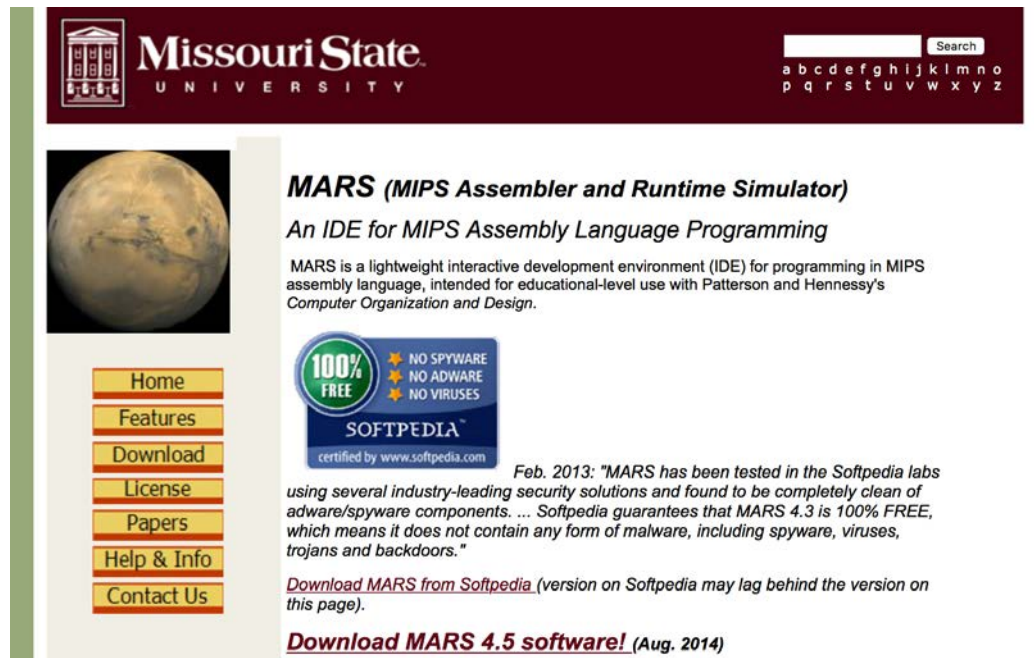
Arquitectura MIPS

- **M**icroprocessor without **I**nterlocked **P**ipeline **S**tages
- 1981 Universidad de Stanford – John L. Hennessy
- Introduce la **ejecución segmentada** de instrucciones en etapas



MARS: MIPS Assembler and Runtime Simulator

- <http://courses.missouristate.edu/KenVollmar/MARS/index.htm>
- Requiere tener instalado www.java.com



The screenshot shows the MARS website with a dark red header. The header contains the Missouri State University logo and a search bar. Below the header, there is a sidebar with a navigation menu: Home, Features, Download, License, Papers, Help & Info, and Contact Us. The main content area features a large image of Mars, the title "MARS (MIPS Assembler and Runtime Simulator)", and a subtitle "An IDE for MIPS Assembly Language Programming". A paragraph describes MARS as a lightweight interactive development environment (IDE) for programming in MIPS assembly language. A Softpedia badge indicates "100% FREE" and "NO SPYWARE, NO ADWARE, NO VIRUSES". A quote from Softpedia dated Feb. 2013 states that MARS has been tested and found to be completely clean of malware. At the bottom, there are links to "Download MARS from Softpedia" and "Download MARS 4.5 software! (Aug. 2014)".

Missouri State UNIVERSITY

Search

abcdefghijklmnopqrstuvwxyz

MARS (MIPS Assembler and Runtime Simulator)

An IDE for MIPS Assembly Language Programming

MARS is a lightweight interactive development environment (IDE) for programming in MIPS assembly language, intended for educational-level use with Patterson and Hennessy's *Computer Organization and Design*.

100% FREE
NO SPYWARE
NO ADWARE
NO VIRUSES
SOFTPEDIA™
certified by www.softpedia.com

Feb. 2013: "MARS has been tested in the Softpedia labs using several industry-leading security solutions and found to be completely clean of adware/spyware components. ... Softpedia guarantees that MARS 4.3 is 100% FREE, which means it does not contain any form of malware, including spyware, viruses, trojans and backdoors."

[Download MARS from Softpedia](#) (version on Softpedia may lag behind the version on this page).

[Download MARS 4.5 software!](#) (Aug. 2014)

Arquitecturas RISC vs CISC

- **CISC**

- *Complex Instruction Set Computer*
- Instrucciones de **longitud variable**
- Repertorio de instrucciones amplio
- Hay instrucciones muy diversas, que nos permiten hacer una tarea compleja con una única instrucción
- **Intel 8086**, Motorola 68K, DEC Vax, DEC Alpha

- **RISC**

- *Reduced Instruction Set Computer*
- Instrucciones de **tamaño fijo**
- Repertorio de instrucciones reducido
- Para hacer una tarea compleja son necesarias varias instrucciones
- **MIPS, ARM**, IBM 360/370, Sun Sparc, HP PA

¿Qué programa está escrito para una arquitectura RISC y cuál para CISC?



Contenidos

- Evolución hasta arquitecturas de microprocesador
- Generación de código máquina
- Practiquemos código binario y hexadecimal
- Conclusiones y bibliografía

¿Cómo se puede generar el código máquina?

- Mediante la compilación de código escrito en lenguaje C
- Tras la fase de enlazado de código C con librerías precompiladas
- Mediante el ensamblado de código escrito en ensamblador
- Manualmente

Código máquina

- Programa en lenguaje de alto nivel (C)

```
swap (int v[], int k){  
    int temp;  
    temp = v[k];  
    v[k] = v[k+1];  
    v[k+1] = temp;  
}
```

Uno-a-muchos

Compilador C

- Programa en lenguaje ensamblador (MIPS)

```
swap:    sll    $2, $5, 2  
         add    $2, $4, $2  
         lw     $15, 0($2)  
         lw     $16, 4($2)  
         sw     $16, 0($2)  
         sw     $15, 4($2)  
         jr     $31
```

Uno-a-uno

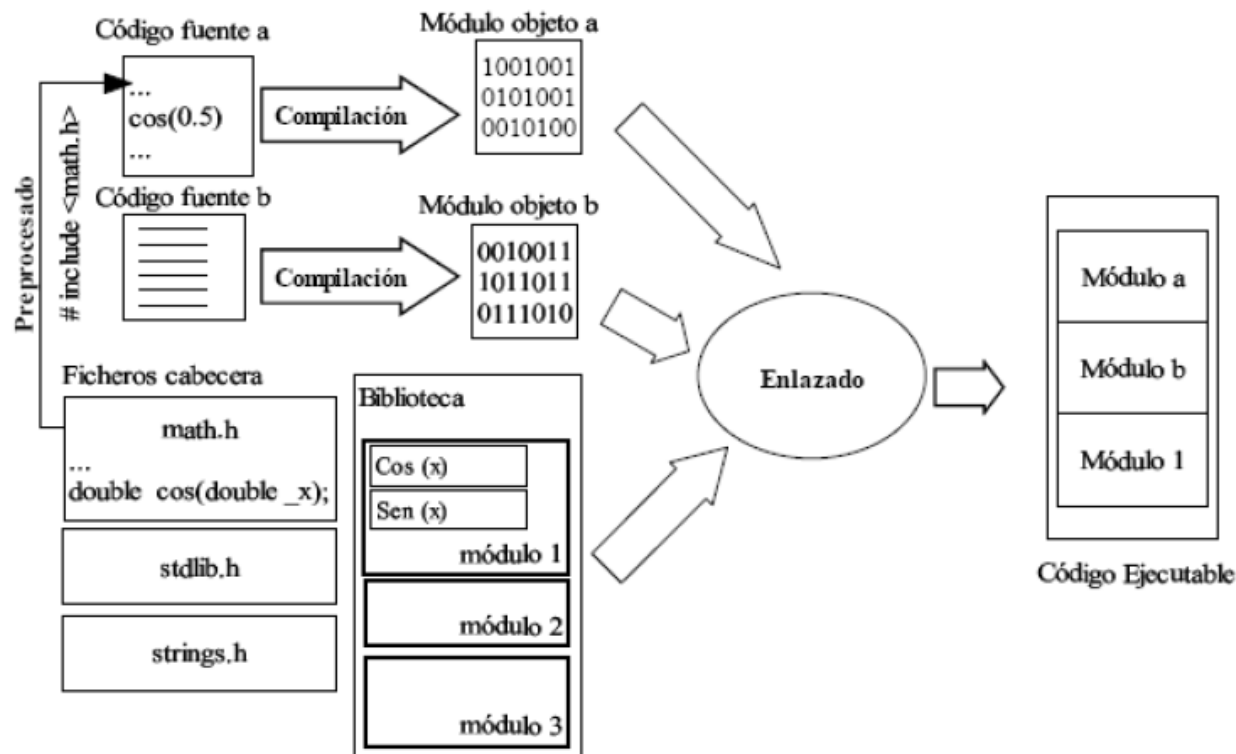
Ensamblador

- Código máquina (object, binario) code (MIPS)

```
000000 00000 00101 0001000010000000  
000000 00100 00010 0001000000100000  
. . .
```


Fases de compilación

- Preprocesado → compilado → enlazado
 - La etapa de **compilación** = compilado + ensamblado



¿Cómo se puede generar el código máquina?

- Mediante la compilación de código escrito en lenguaje C
- Tras la fase de enlazado de código C con librerías precompiladas
- Mediante el ensamblado de código escrito en ensamblador
- Manualmente

Compilación de programas C en Linux

- Compilador gcc (GNU C/C++ compiler)
 - **gcc [opciones] archivos**
 - **archivos** son archivos de código fuente, código objeto, etc.
 - **opciones** básicas:
 - -o nombre: pone nombre al ejecutable generado
 - -E: realizar el preprocesado, pero no compilar
 - gcc -E archivo.c > archivoprep
 - -S: realizar la compilación, pero no ensamblar
 - gcc -S archivo.c -o archivo.s
 - -c: suprimir etapa de enlazado y generar archivos objeto (extensión .o)
 - gcc -c archivo.c -o archivo.o
 - -lbib: enlaza con la biblioteca de nombre libbib.a

Compilación de programas C en Linux

- Ejemplos de utilización convencional:
 - gcc programa.c
 - Genera ejecutable a.out
 - gcc programa.c -o miprograma
 - Genera ejecutable *miprograma*
 - gcc archivo1.c archivo2.c -o miprograma
 - Combina códigos objetos generados en un ejecutable *miprograma*
 - **Esta forma de combinar varios fuentes no es la óptima a la hora de compilar: usar makefiles**

Ejercicio: comprobando las etapas de compilación

- Escribe un archivo simple de código en C y genera:
 - El archivo ejecutable
 - Los archivos intermedios de compilación
- Muestra por consola el contenido de los archivos
 - En linux puedes usar cat o more
 - ¿Qué tipo de código es el de la imagen de la derecha?
 - ¿De qué arquitectura?

```
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq    $16, %rsp
    leaq    L_.str(%rip), %rdi
    movl    $0, -4(%rbp)
    movl    $5, -8(%rbp)
    movl    -8(%rbp), %esi
    movb    $0, %al
    callq   _printf
    xorl    %esi, %esi
    movl    %eax, -12(%rbp)    ## 4-byte Spill
    movl    %esi, %eax
    addq    $16, %rsp
    popq    %rbp
    retq
    .cfi_endproc

    .section      __TEXT,__cstring,cstring_literals
L_.str:
    .asciz      "\nHola Mundo, %d\n"

.subsections_via_symbols
```

Contenidos

- Evolución hasta arquitecturas de microprocesador
- Generación de código máquina
- Practiquemos código binario y hexadecimal
- Conclusiones y consulta adicional

Bits, bytes y palabras

- Bit: unidad binaria de información
 - Puede valer 0 ó 1
- Byte: conjunto de 8 bits
 - Permite representar 2^8 valores: desde 0 hasta 2^8-1
- Tamaño de la palabra de datos (word)
 - Depende de la arquitectura: 32 bits vs 64 bits

Decimal	Binario
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Código binario

- Código binario: representación en base 2

- Valores dígitos < 2

- Ej. $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$

- En una representación en base p

- Valores dígitos $< p$

- $3276_p = 3 \times p^3 + 2 \times p^2 + 7 \times p^1 + 6 \times p^0$

- Ej. Suponer $p = 10$

- $3276_{10} = 3 \times 10^3 + 2 \times 10^2 + 7 \times 10^1 + 6 \times 10^0$

- Ej. Suponer $p = 8$

- $3276_8 = 3 \times 8^3 + 2 \times 8^2 + 7 \times 8^1 + 6 \times 8^0 = 1726_{10}$

Datos e instrucciones en código binario

- Almacenamiento de datos en memoria principal / registros
 - En una arquitectura de 32 bits el valor 19 en memoria se almacena
0000000000000000000000000000000010011
 - No confundir Bytes/KBytes/MBytes con bits/Kbits/Mbits
 - El dato anterior ocupa 32 bits o 4 bytes
- Almacenamiento de instrucciones
 - En arquitectura MIPS de 32 bits, la instrucción sub \$s0,\$s1,\$s2 se almacena
00000010001100101000000000100010
 - Longitud de la instrucción en bits según arquitecturas de 32 bits vs. 64 bits

Representación hexadecimal

- Expresar de forma más compacta **datos y direcciones** de memoria
- Símbolos con valores de 0 a 15
 - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- Muy fácil pasar de binario a hexadecimal
 - Grupo de 4 bits → Un símbolo hexadecimal
 - Palabra de 32 bits → 8 símbolos hexadecimal

0000	0010	0011	1101	1000	0000	1111	0010
↓	↓	↓	↓	↓	↓	↓	↓
0	2	3	D	8	0	F	2
0x023D80F2							

Traduce entre códigos

Decimal	Binario	Hexadecimal
24		
	011100	
		0x1F
		0x20

Traduce entre códigos

Decimal	Binario	Hexadecimal
24		
	011100	
		0x1F
		0x20

Representación de números negativos en complemento a 2

	Binario C2	Decimal	Hex
	1000	-8	0x8
	1001	-7	0x9
	1010	-6	0xA
	1011	-5	0xB
	1100	-4	0xC
	1101	-3	0xD
	1110	-2	0xE
	1111	-1	0xF
	0000	0	0x0
	0001	1	0x1
	0010	2	0x2
	0011	3	0x3
	0100	4	0x4
	0101	5	0x5
	0110	6	0x6
	0111	7	0x7

Diagram illustrating the 2's complement representation of negative numbers:

- Complementar** (Green arrow): Shows the process of taking the complement of a positive number. For example, 0101 (5) is complemented to 1011 (-5).
- Sumar 1** (Blue arrow): Shows the process of adding 1 to the complemented number. For example, 1011 (-5) is incremented to 1010 (-4).
- Complementar** (Blue arrow): Shows the process of taking the complement of a negative number. For example, 1010 (-4) is complemented to 0101 (5).
- Sumar 1** (Green arrow): Shows the process of adding 1 to the complemented number. For example, 0101 (5) is incremented to 0110 (6).

Final results shown in green:

- 0101 (5) is complemented to 1011 (-5).
- 1011 (-5) is incremented to 1010 (-4).
- 1010 (-4) is complemented to 0101 (5).
- 0101 (5) is incremented to 0110 (6).

Contenidos

- Evolución hasta arquitecturas de microprocesador
- Generación de código máquina
- Practiquemos código binario y hexadecimal
- Conclusiones y consulta adicional

Conclusiones

- Evolución exponencial capacidad de procesamiento
 - Ley de Moore y Ley de rendimientos acelerados
- Arquitecturas de procesador
 - Von Neumann (microprocesador) vs Harvard (microcontroladores, MIPS)
 - RISC vs CISC
- Generación de código máquina
 - Fases de compilación
 - Conocer cómo traducir entre binario y hexadecimal
 - Números negativos

Consulta adicional

- Historia de la informática (Paul E. Ceruzzi)
 - PDF accesible en el campus virtual
 - [Historia de las computadoras](#)
- [Nuevo procesador M1 de Apple](#)
- Conversión entre decimal y binario
 - [De binario a decimal](#)
 - [De binario a hexadecimal](#)
- Estructura y Diseño de Computadores
 - Sección 2.4: Números con signo y sin signo