

Arquitectura x86

IIC2343 - Arquitectura de Computadores

Nicolás Elliott B. (nicolas.elliott@uc.cl)



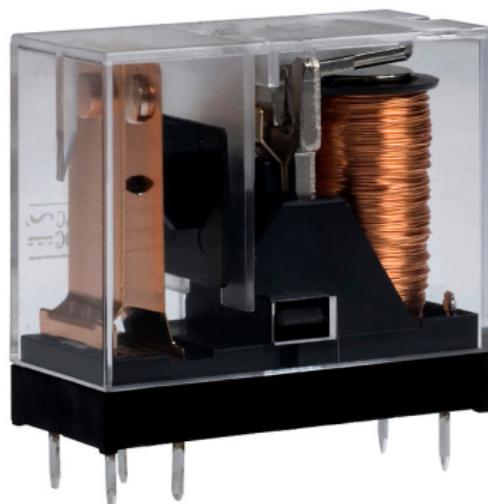
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
ESCUELA DE INGENIERÍA
PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

(II/2019)

Historia

Generación 0 (≈1830)

Relés

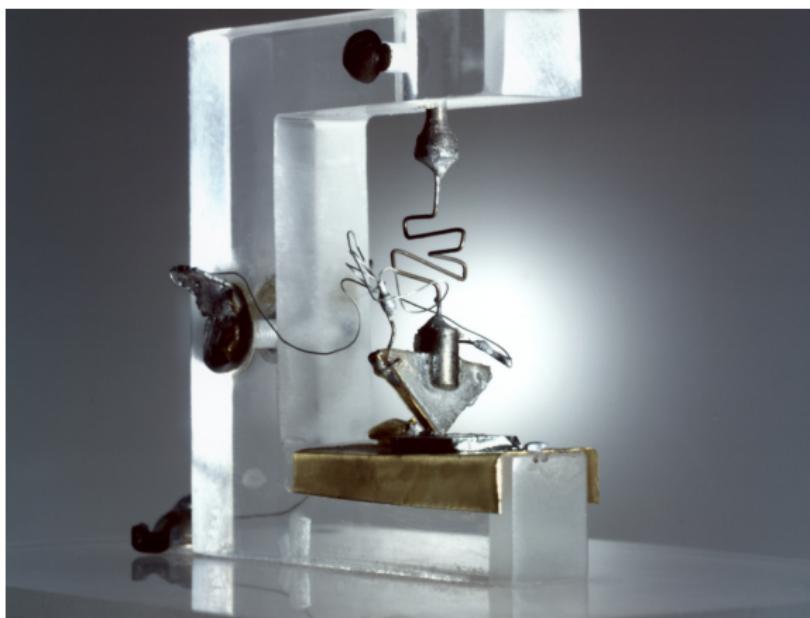


Generación 1 (≈ 1900)

Tubos



Generación 2



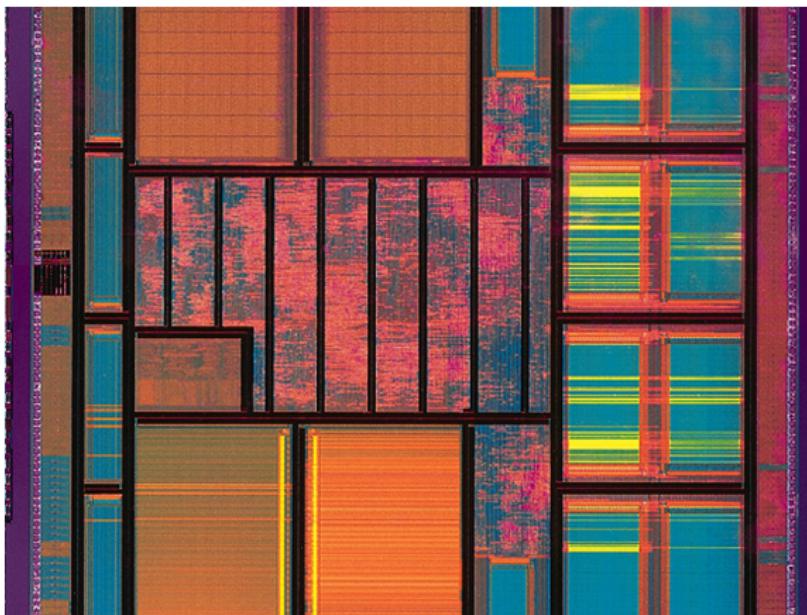
Generación 2 (1947)

Transistores



Generación 3 (1958)

Transistores

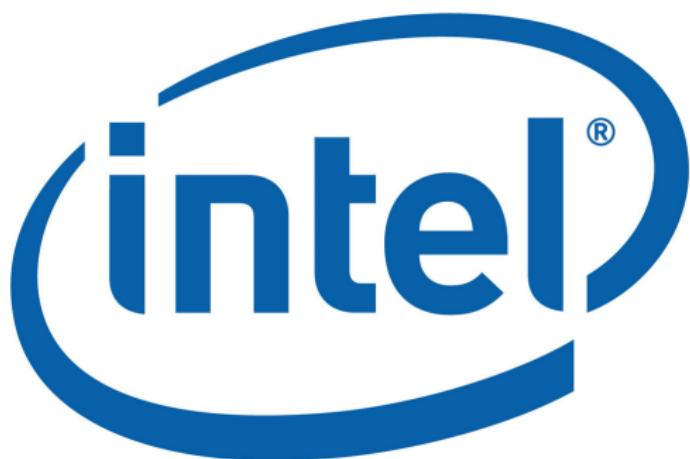


Generación 3 (1958)

Gordon Moore y Robert Nyce

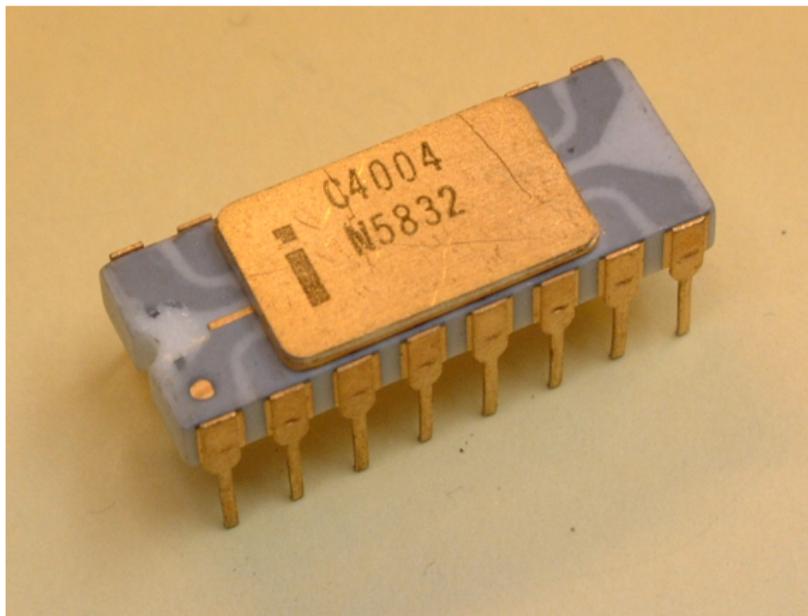


Intel (1968)



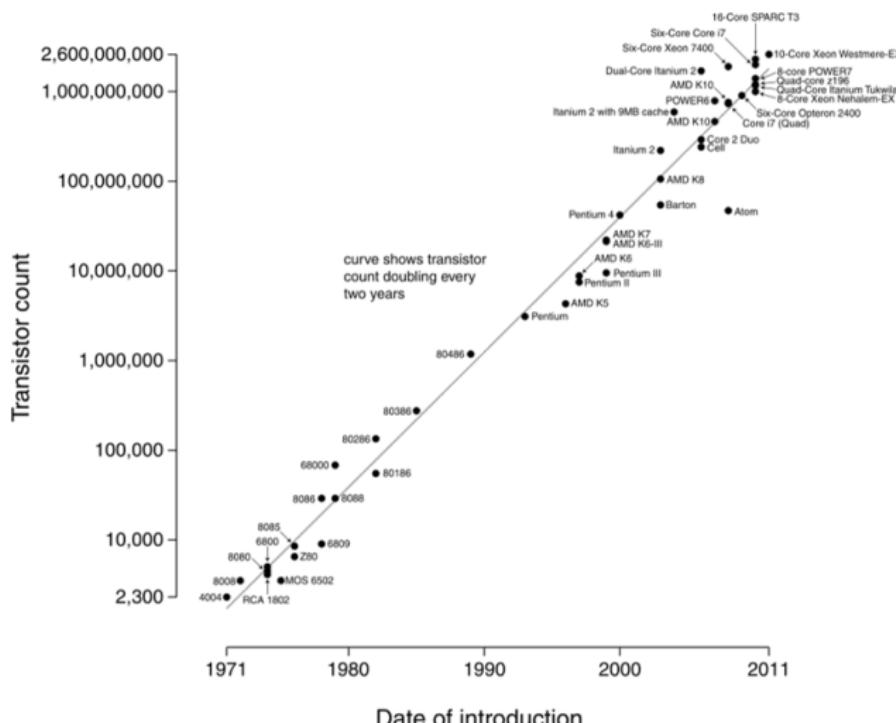
Intel

Intel 4004 (1971)



Ley de Moore

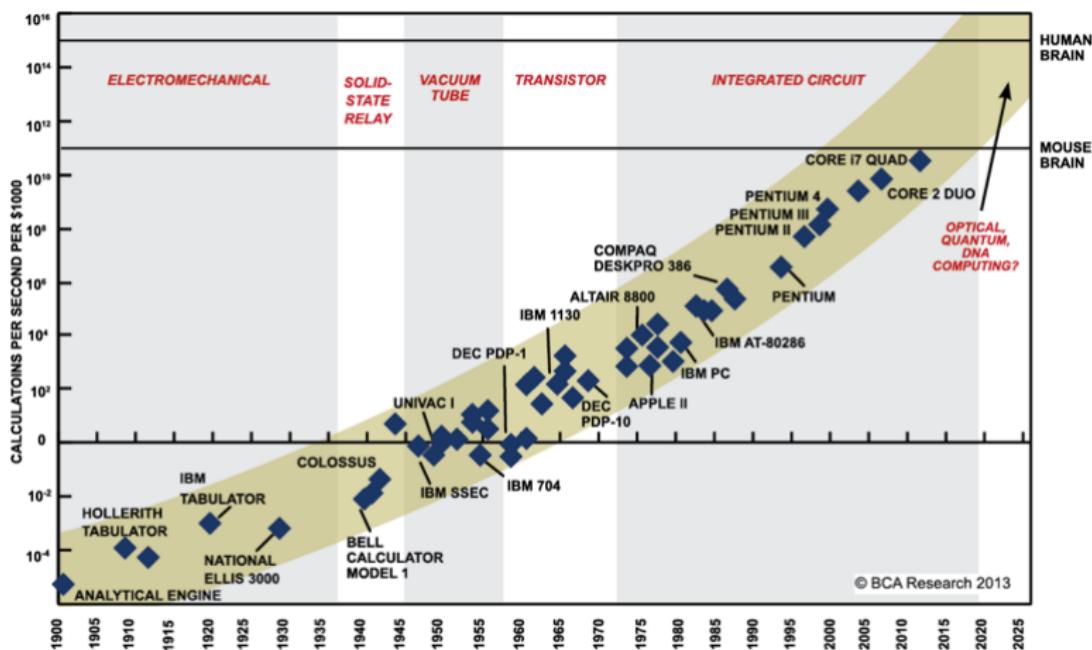
Microprocessor Transistor Counts 1971-2011 & Moore's Law



Ley de Moore

https://www.youtube.com/watch?v=7uvUiq_jTLM

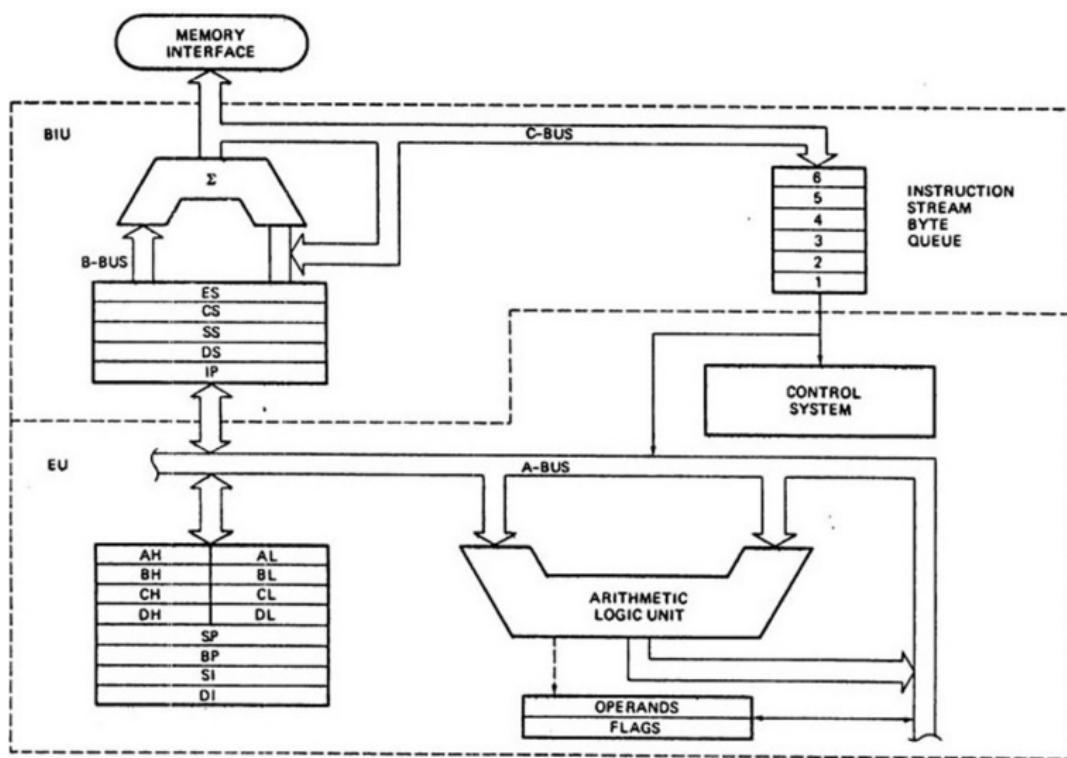
Singularidad



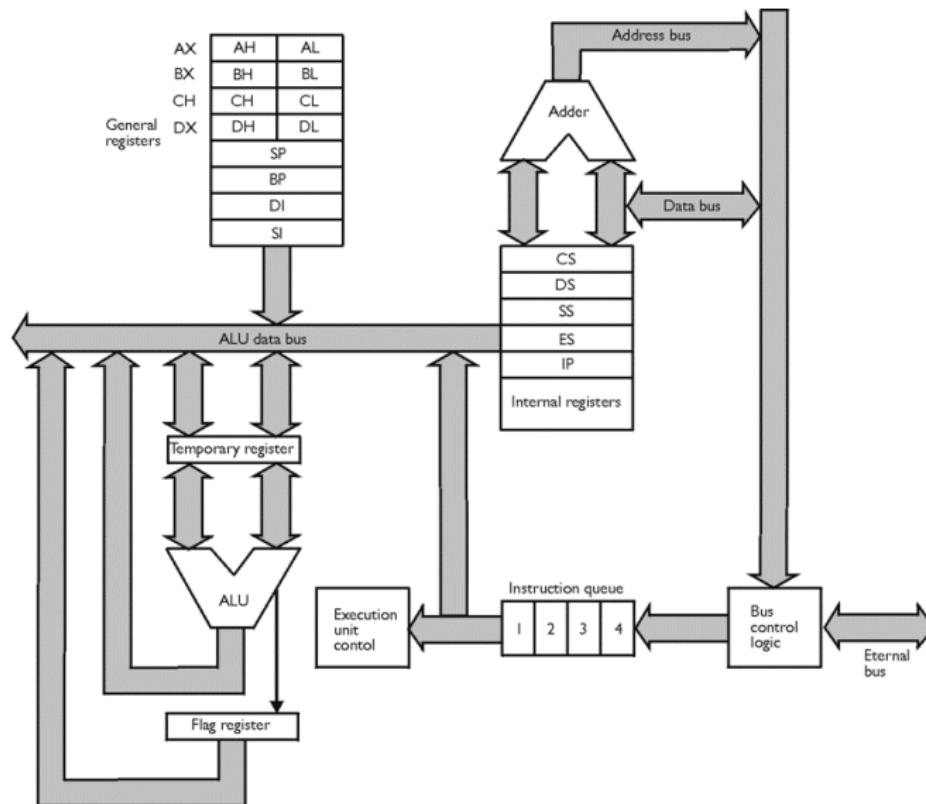
SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, THE VIKING PRESS, 2006. DATAPoints BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

x86

Intel 8086 (1978)

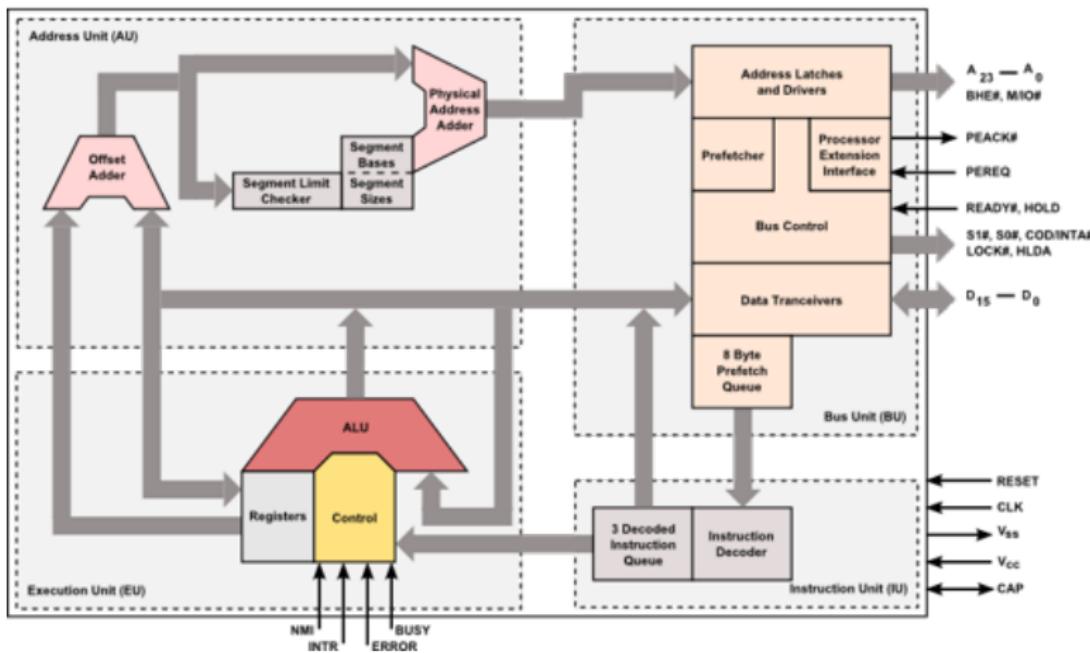


Intel 80186 (1982)

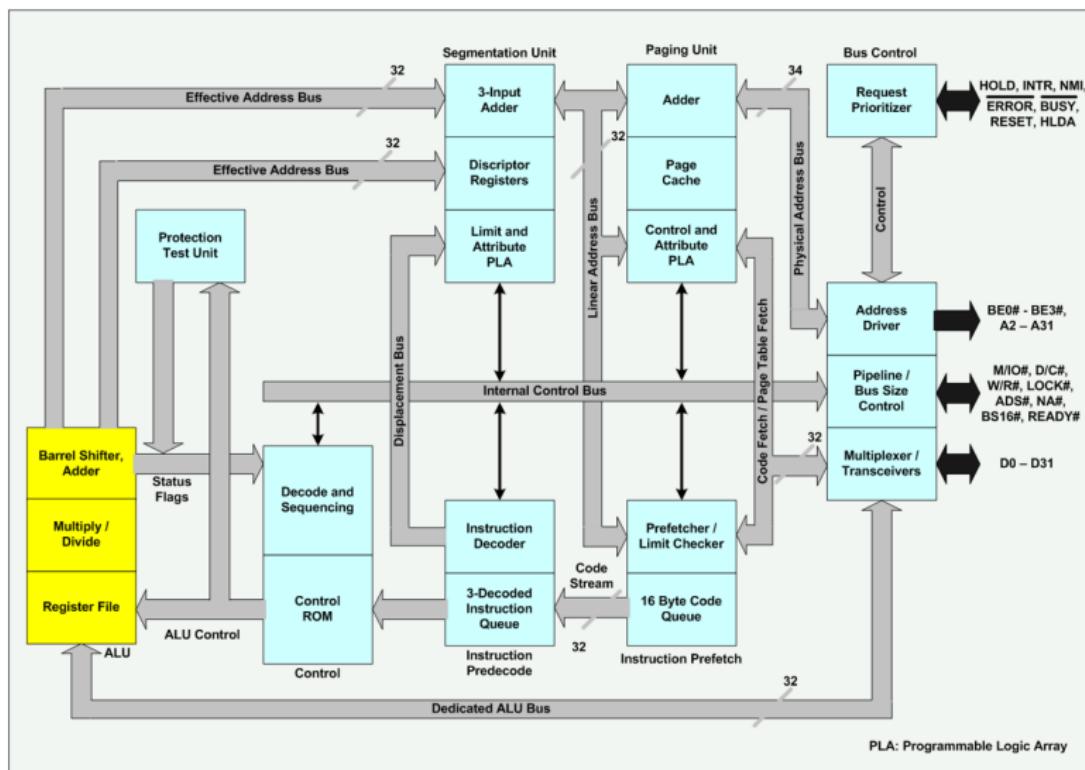


Intel 80286 (1982)

Intel 80286 architecture

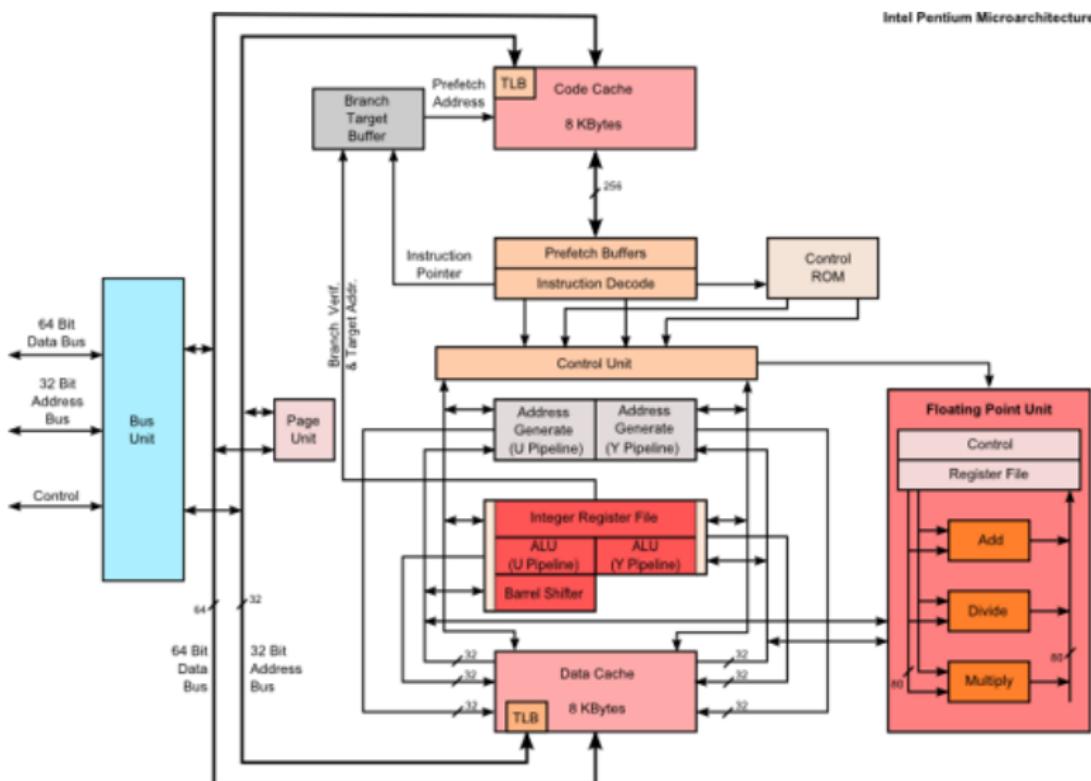


Intel 80386 (1985)



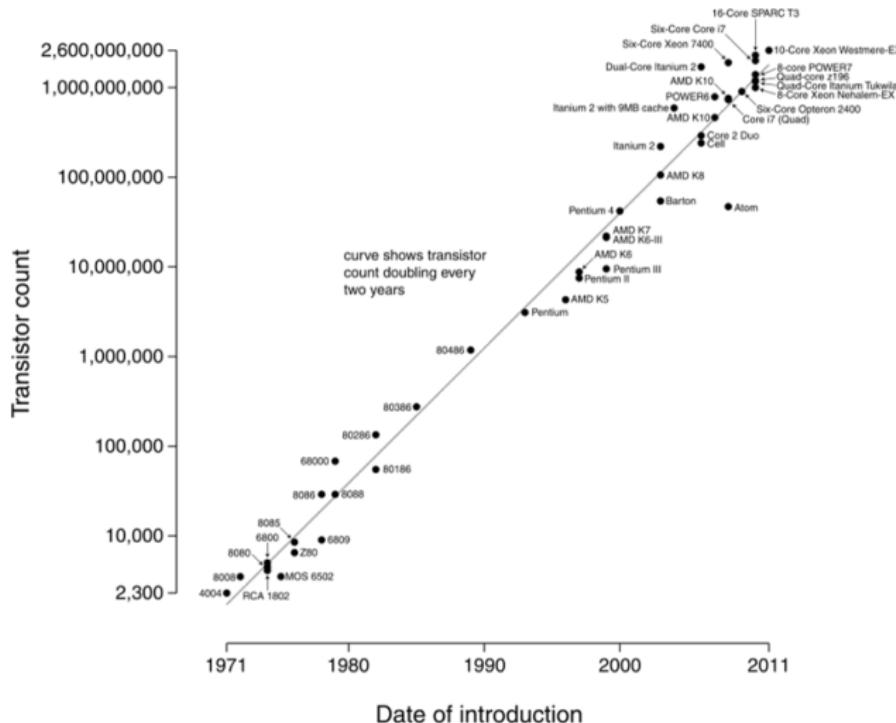
PLA: Programmable Logic Array

Intel P5 Pentium (1993)



Ley de Moore

Microprocessor Transistor Counts 1971-2011 & Moore's Law



x86

Registros

- 4 registros de 16 bits con propósito general (AX, BX, CX, DX). Divisibles en sectores altos y bajos (AX = AH | AL).
- BX se utiliza además para el direccionamiento indirecto (registro base)
- 2 registros de 16 bits para el uso general y direccionamiento indirecto (SI, DI), usados como registro índice.
- Instruction pointer (IP), stack pointer (SP) y base pointer (BP), todos de 16 bits.

Microarquitectura

- unidad de control microcode (CISC).
- Sólo una alu como unidad de ejecución (FPU se introdujo en el 486)
- 6 condition codes: Z, S, C ,O ,P y A
- Direccionamiento de memoria de 16 bits.
- Palabras en memoria de 8 bits.
- Stack en memoria, SP apunta al **último elemento ingresado en el stack**.

ISA

- Instrucciones de transferencia, aritméticas, lógicas, saltos, subrutinas.
- Tipos de datos nativos de 8 y 16 bits, con y sin signo.
- Múltiples tipos de direccionamiento:
 - directo
 - indirecto por reg.
 - indirecto por reg. base y offset
 - indirecto con reg. base y reg. Índice
 - indirecto con reg. base, reg. índice y offset

Variables

- Soporta 2 tipos: byte de 8 bits y word de 16 bits.
- Son representados por db (byte) y dw (word).
- Arreglos también pueden ser de estos tipos.
- Datos son almacenados en little endian.
- Todo esto implica que manejo de memoria requiere mayor cuidado.
- Instrucción LEA reg, var nos permite almacenar en el registro reg la dirección de la variable var.

Variables

variables: var1 db 0x0A y var2 dw 0x07D0, y dos arreglos: arr1 db 0x01, 0x02, 0x03 y arr2 dw 0x0AOB, 0x0C0D , el estado de la memoria sería el siguiente:

Variable	Dirección (16 bits)	Palabra (8 bits)
var1	100	0x0A
var2	101	0xD0
	102	0x07
arr1	103	0x01
	104	0x02
	105	0x03
arr2	106	0x0B
	107	0x0A
	108	0x0D
	109	0x0C

Multiplicación

Código en C

```
int mult()
{
    int a = 10;
    int b = 200;
    int res = 0;
    while(a > 0)
    {
        res += b;
        a--;
    }
    return res;
}
```

Multiplicación

Código en x86

```
MOV AX, 0
MOV CX, 0
MOV DX, 0
MOV CL, a          ;CL guarda el valor de a
MOV DL, b          ;DL guarda el valor de b

start:
CMP CL, 0          ;IF a <= 0 GOTO end
JLE endprog

ADD AX, DX          ;AX += b
DEC CL              ;a--
JMP start

endprog:
MOV res, AX          ;res = AX

RET

a      db 10
b      db 200
res    dw 0
```

Multiplicación

CISC

- Arquitectura CISC permite incluir instrucciones aritméticas complejas como MUL y DIV, que utilizan varios ciclos:
 - $\text{MUL op} \Rightarrow AX = AL \times op$
 - $\text{MUL op} \Rightarrow DX|AX = AX \times op$
 - $\text{DIV op} \Rightarrow AL = AX \div op$
 - $\text{DIV op} \Rightarrow AX = DX|AX \div op$

Multiplicación

Código en x86 usando MUL

```
;Calculo de la multiplicacion res = a*b

MOV AX, 0

MOV AL, a          ;AL = a
MUL b            ;AX = AL*b

MOV res, AX        ;res = AX

RET

a      db 10
b      db 200
res    dw 0
```

Subrutinas

Computador Básico

- En nuestro computador, el stack almacenaba la dirección de retorno.
- No había una convención sobre donde almacenar los parámetros y valores de retorno.

x86

- En x86, utilizaremos el stack de manera más explícita: parámetros, variables locales.
- Uso del registro BP (base pointer) es fundamental para facilitar el manejo de todos estos datos.

Subrutinas

Convenciones de llamada (calling conventions)

- Las convenciones definen la interfaz sobre la cual trabajará el código de la subrutina.
- Una convención de llamada debe especificar lo siguiente:
 - Donde se encuentran los parámetros (stack, registros o una mezcla de ambos).
 - Si se usa el stack, el orden en que los parámetros son entregados.
 - Definición de responsabilidades de restauración del stack, entre la subrutina y el código que la llama.

Subrutinas

Convenciones de llamada (calling conventions)

- Existen múltiples convenciones: stdcall, cdecl, fastcall, safecall, syscall, thiscall,...
- Se dividen entre las que asignan la responsabilidad de limpieza del stack a la subrutina, y las que se la asignan al código que llama a la subrutina.
- Ocuparemos la convención stdcall, que es la usada por la API Win32 de Microsoft.

Subrutinas

stdcall deposita en la subrutina la responsabilidad de limpiar el stack.

- Los parámetros son pasados de derecha a izquierda, usando el stack.
- El retorno se almacenará en el registro AX.
- La subrutina se debe encargar de dejar SP apuntando en la misma posición que estaba antes de pasar los parámetros.

Subrutinas

En stdcall, SP y BP permiten tener llamadas anidadas de subrutinas (recursión) y variables locales

SP →	Variables locales
BP →	Base Pointer anterior a la llamada Dirección de retorno
	Parámetros de la subrutina

Subrutinas

Se deben ejecutar 2 pasos al momento de llamar a una subrutina

- 1 Agregar los parámetros al stack usando la instrucción PUSH. Los valores agregados sólo pueden ser de 16 bits.
- 2 Llamar a la subrutina con la instrucción CALL, lo que almacena en el stack la dirección de retorno y ejecuta el salto a la dirección de la subrutina.

Subrutinas

Dentro de la subrutina, son 5 los pasos a ejecutar

- 1 Guardar el valor actual de BP en el stack y cargar el valor de SP en BP:

PUSH BP

MOV BP, SP

En caso de usar variables locales, se debe reservar espacio para estas, moviendo SP n posiciones hacia arriba, donde n es el número de palabras de memoria que usan las variables:

SUB SP, n

Subrutinas

Dentro de la subrutina, son 5 los pasos a ejecutar

- 2 Ejecutar la subrutina. Para acceder a los parámetros se usa direccionamiento mediante BP. Así, el primer parámetro estará en BP+4, el segundo en BP+6, etc. De la misma manera, las variables locales se acceden usando BP, pero con offset negativo, BP-2, etc
- 3 Al finalizar la subrutina se debe recuperar el espacio de las variables locales:

ADD SP, n

Subrutinas

Dentro de la subrutina, son 5 los pasos a ejecutar

- 4 Rescatar el valor previo de BP:

POP BP

- 5 Mover SP al valor previo al paso de los parámetros:

RET n

donde n indica la cantidad de palabras de memoria usadas por los parámetros.

Subrutinas

Factorial en C

```
int factorial(int n)
{
    if(n==0)
        return 1;
    return n*factorial(n-1);
}

void main()
{
    int n = r
    int fact = factorial(n);
}
```

Recordatorio stdcall

- Convención stdcall:
 - 1 Parámetros de derecha a izquierda.
 - 2 Resultado en registro AX.
 - 3 Debe dejar SP igual que antes de los parámetros.
- Al empezar y durante la subrutina:
 - 1 Guardar el valor actual de BP en el stack y cargar el valor de SP en BP.
 - 2 Reservar espacio en el stack para variables locales.
 - 3 Acceder a parámetros y variables locales mediante BP y offsets positivos y negativos.
- Al finalizar la subrutina:
 - 1 Recuperar espacio usado por las variables locales.
 - 2 Rescatar el valor previo de BP.
 - 3 Setear SP al valor previo al paso de los parámetros.