Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

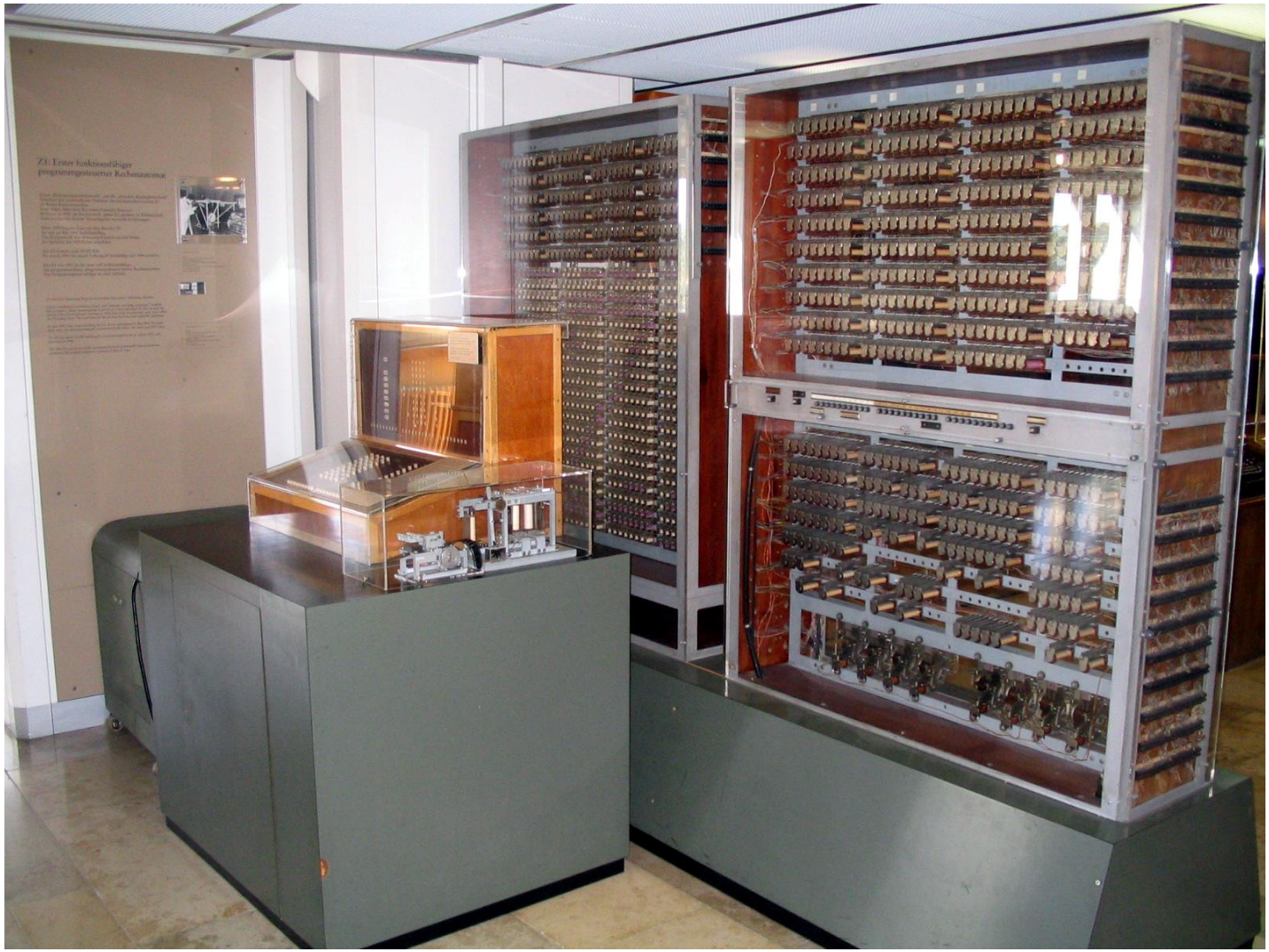
IIC2343 – Arquitectura de Computadores

Arquitectura x86 – Elementos Básicos

Profesor: Hans Löbel

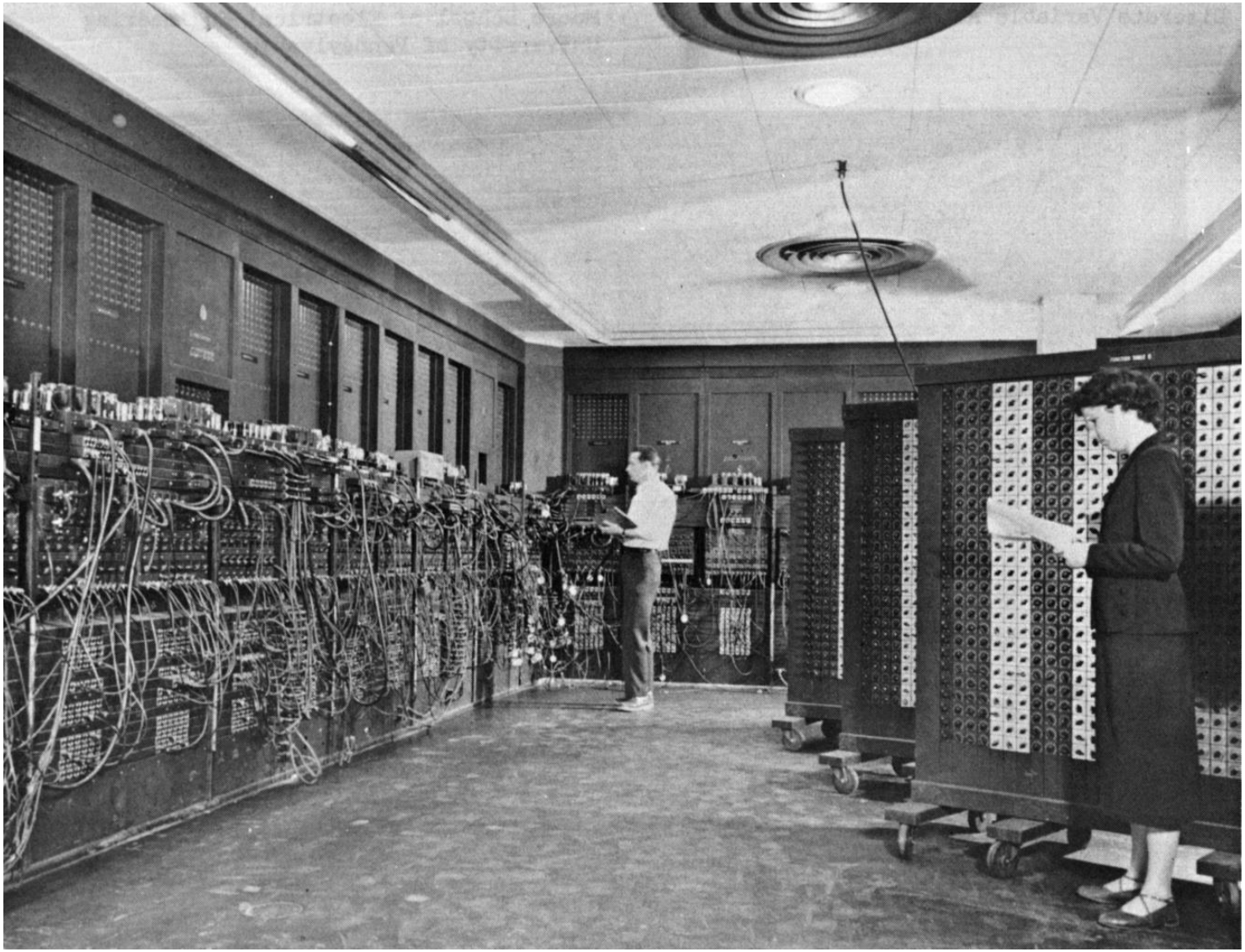
Veamos un poco de historia
(de la computación digital)

GENERACIÓN 0 (≈1830)


Z3: Erster funktionsfähiger
programmgesteuerter Rechenautomat

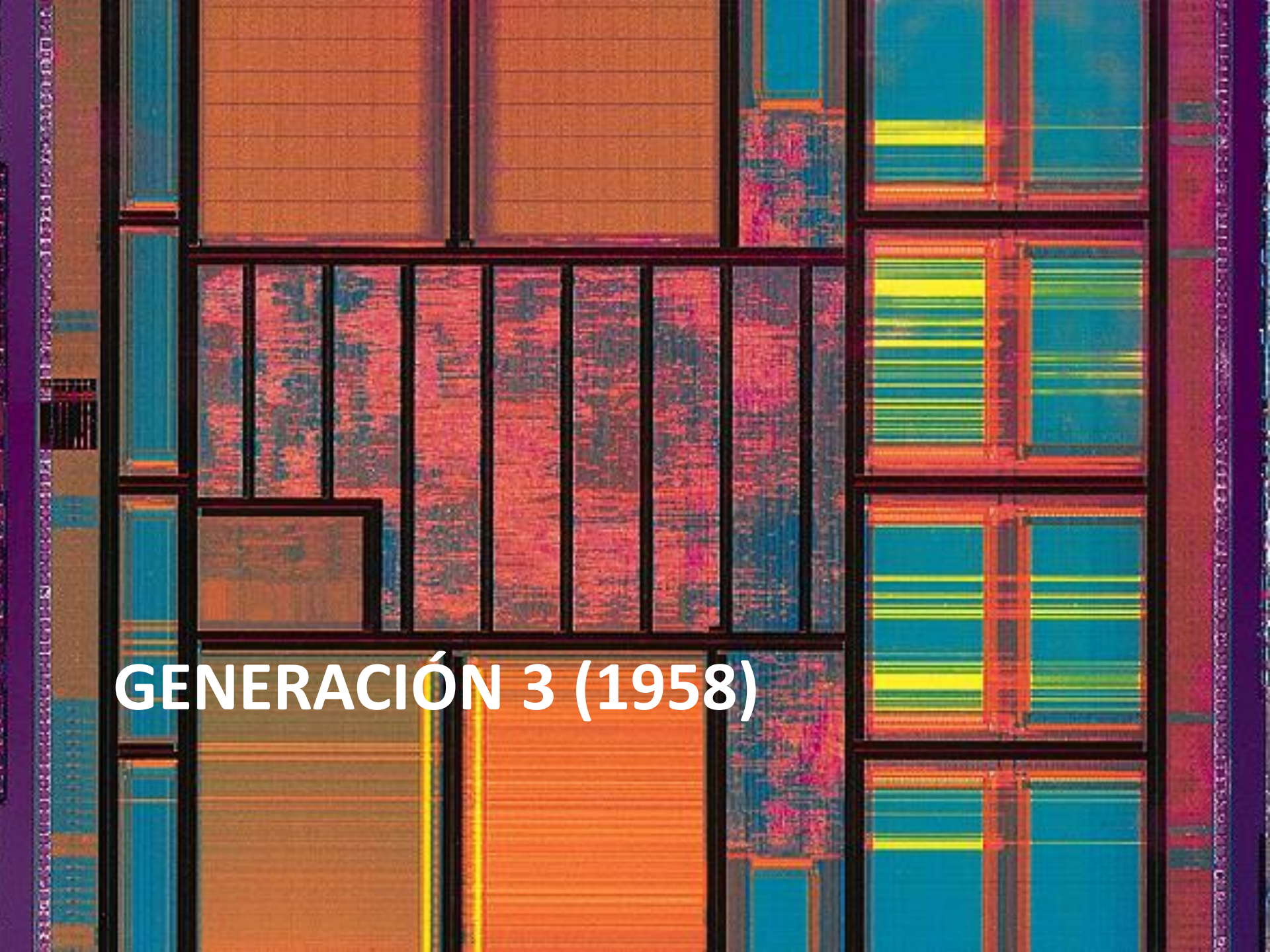
Z3 (1941)

GENERACIÓN 1 (≈1900)



ENIAC (1945)

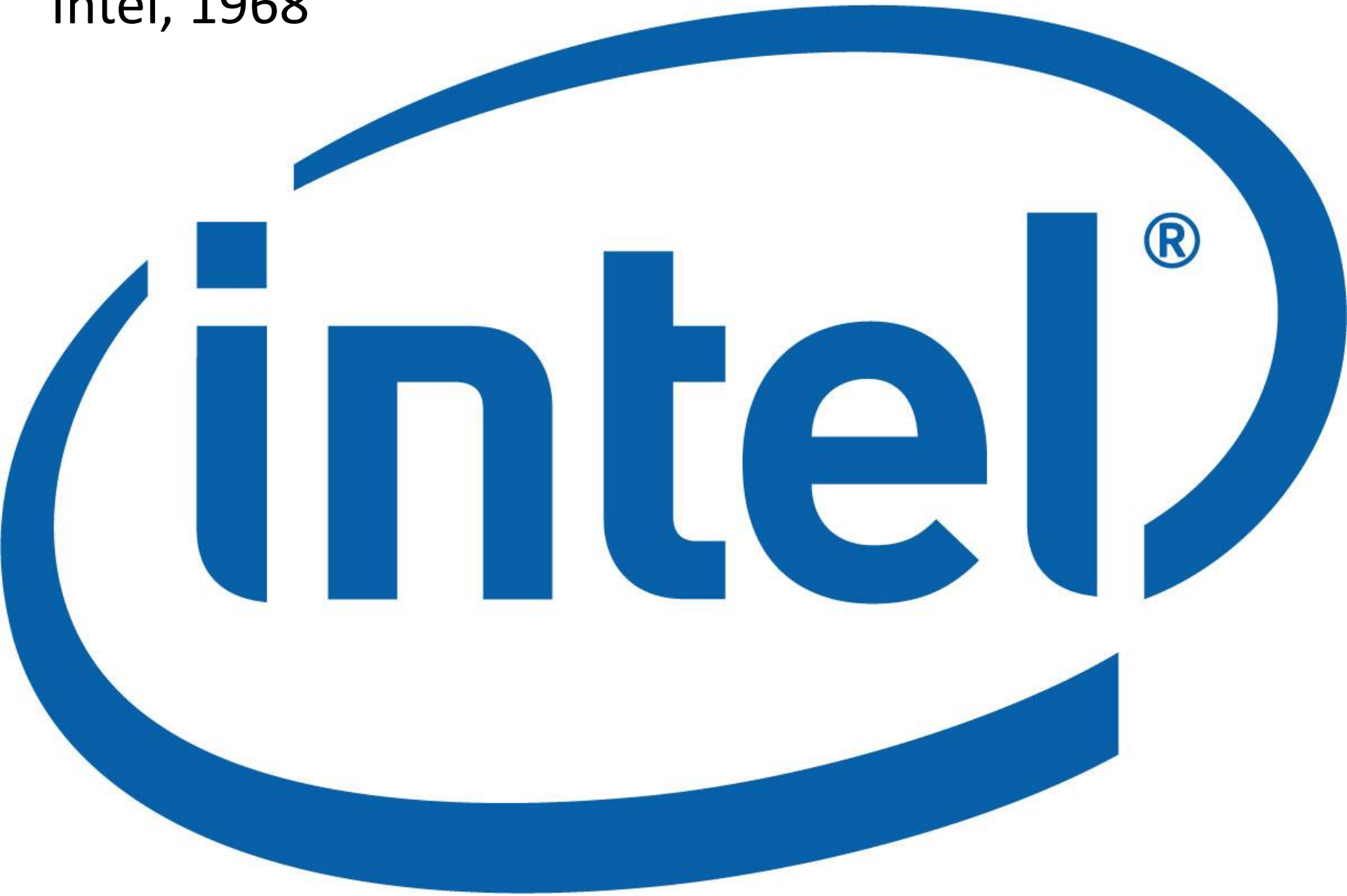
GENERACIÓN 2 (1947)



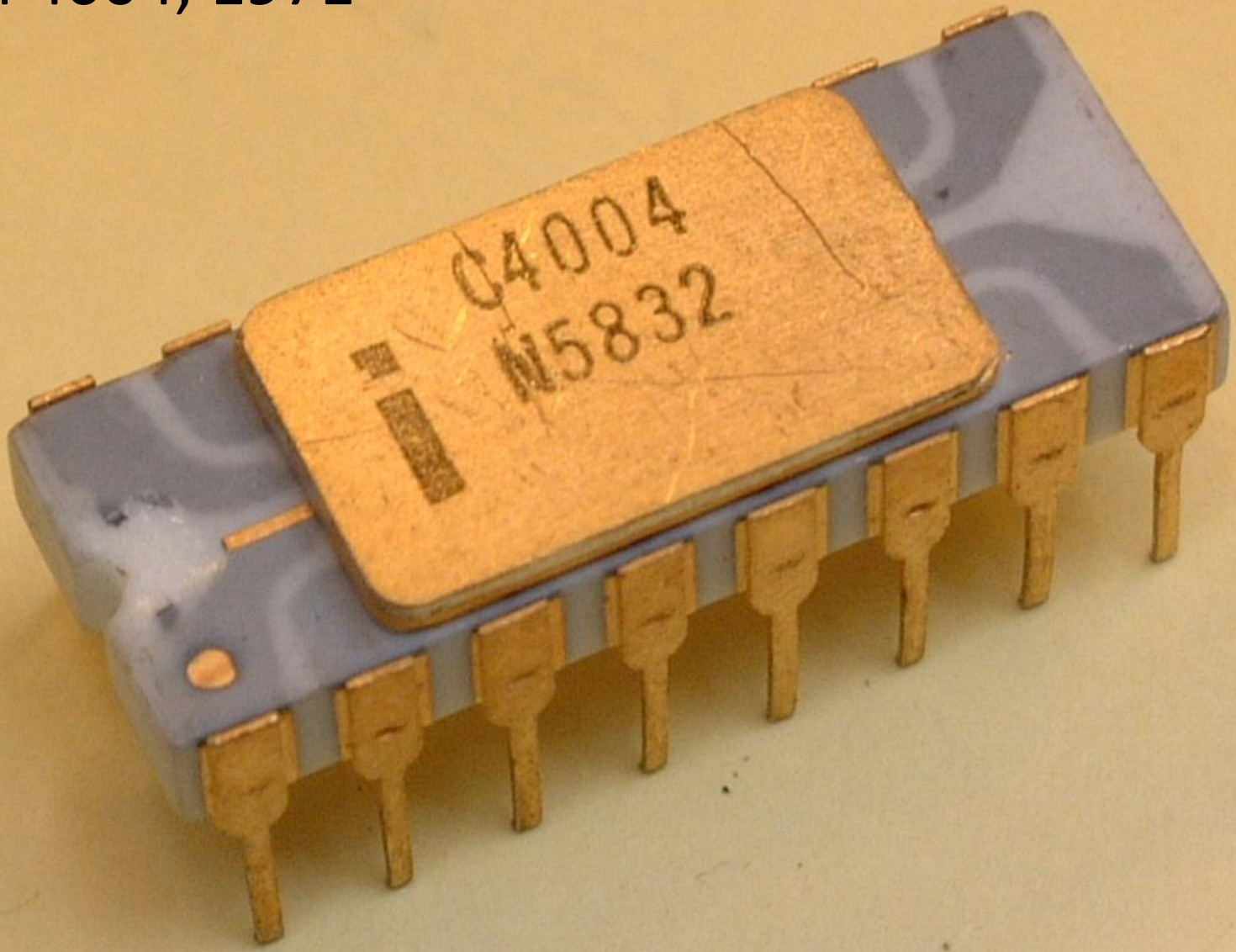
GENERACIÓN 3 (1958)


Gordon
Moore
Robert
Noyce

Intel, 1968



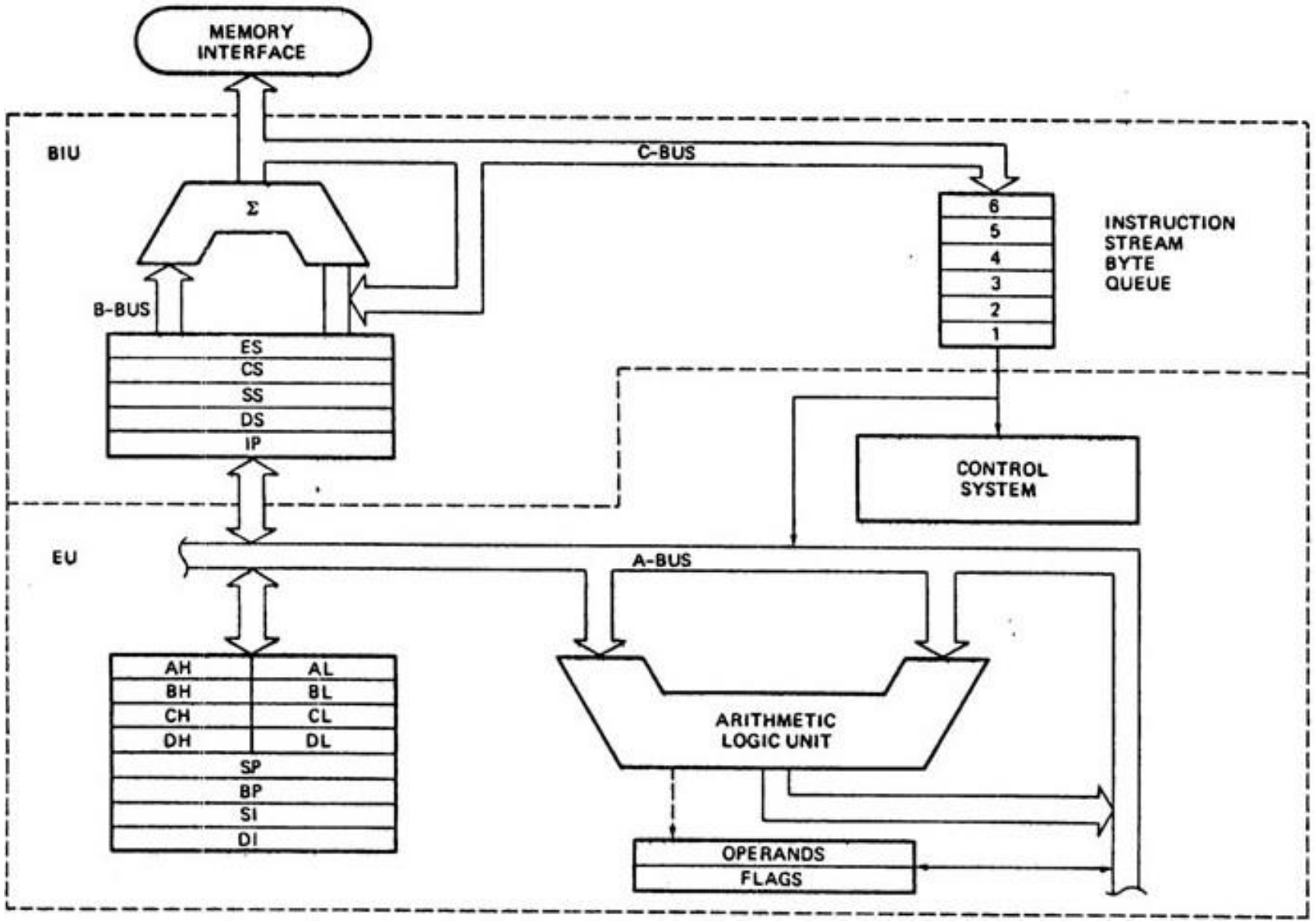
Intel 4004, 1971



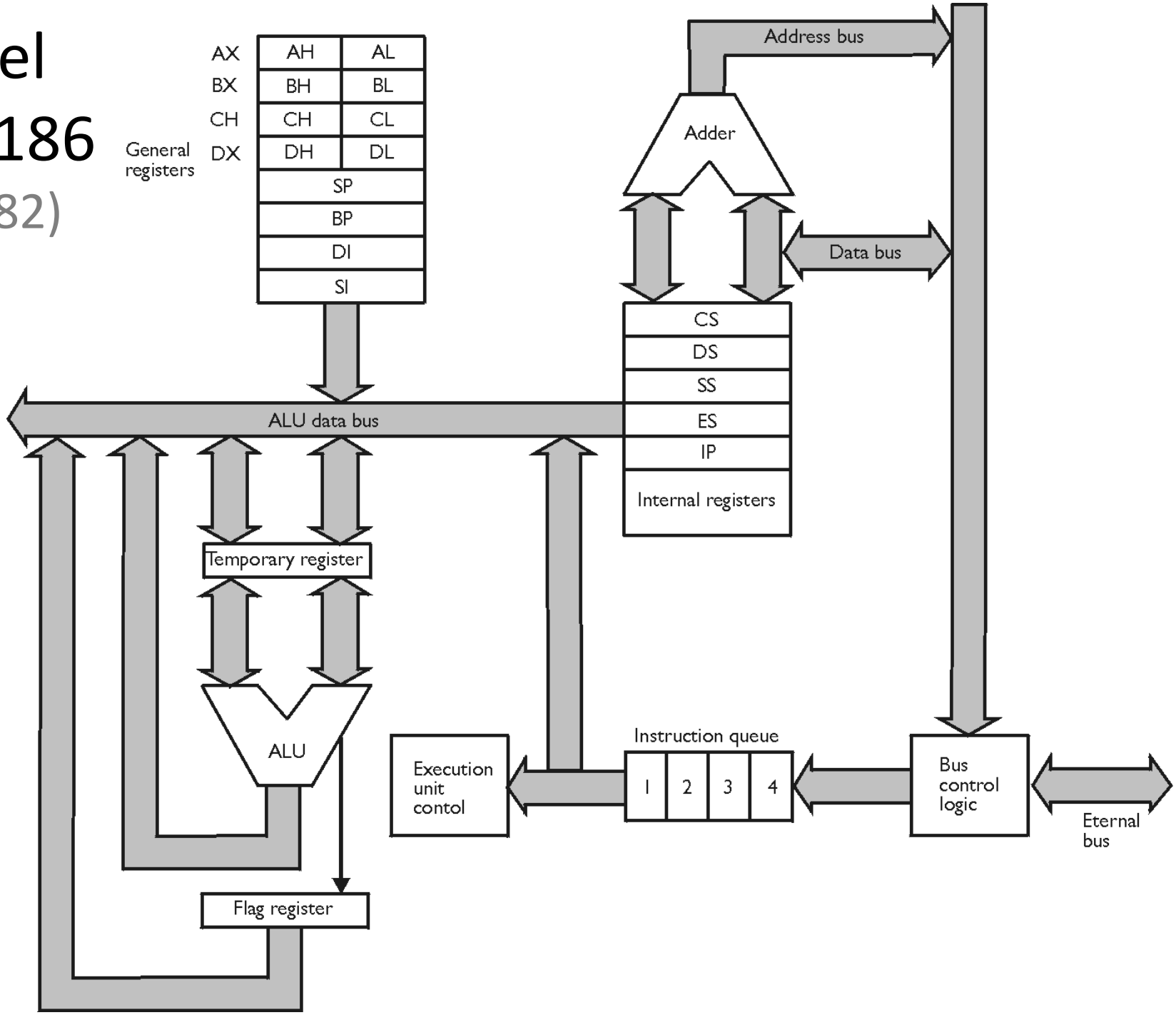
Arquitectura x86 es de las más utilizadas en la actualidad

- El conocimiento de esta arquitectura resulta fundamental para aplicaciones complejas.
- Presenta algunas diferencias claves con la arquitectura del computadores básico.

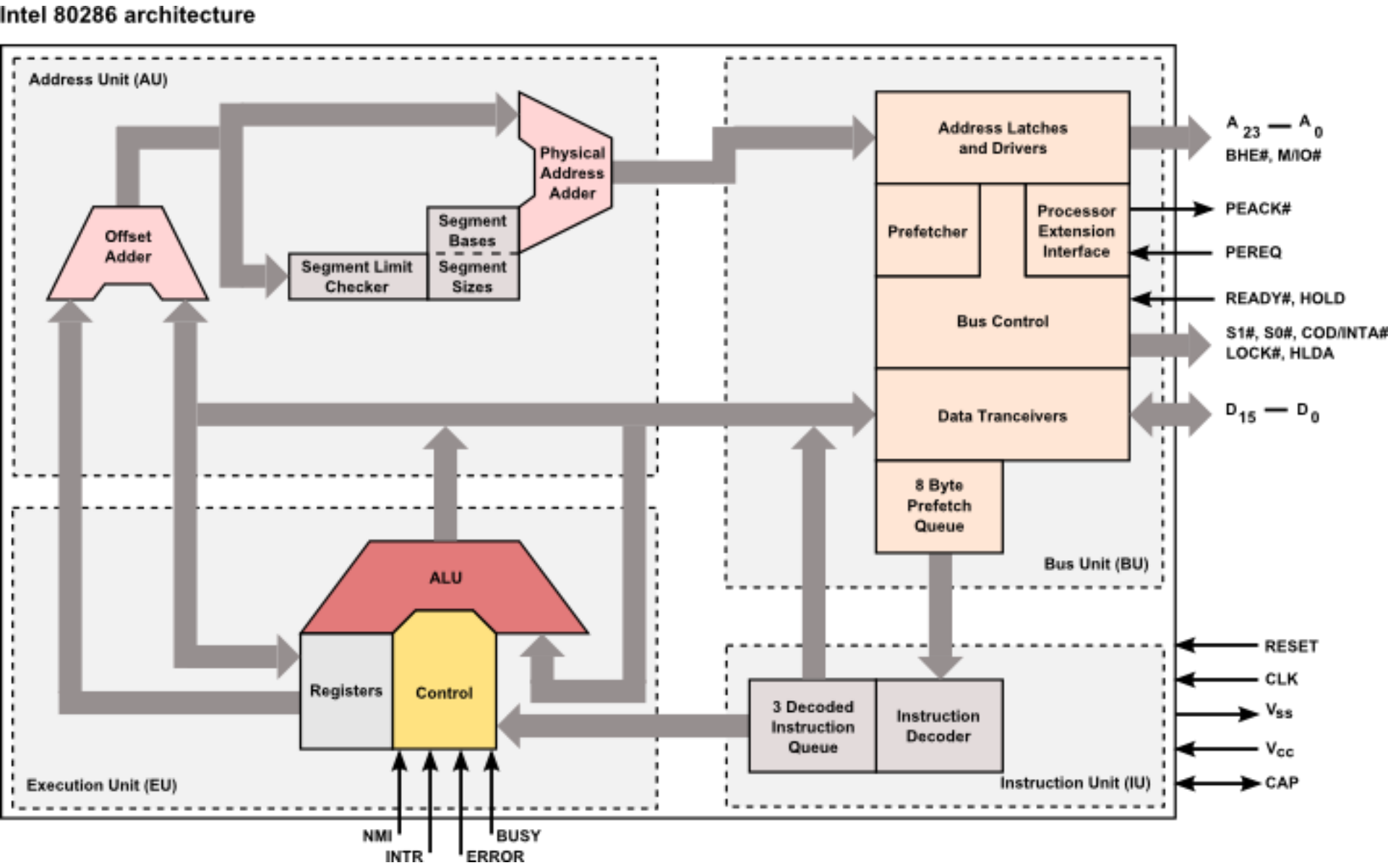
Intel 8086 (1978): inicio del x86



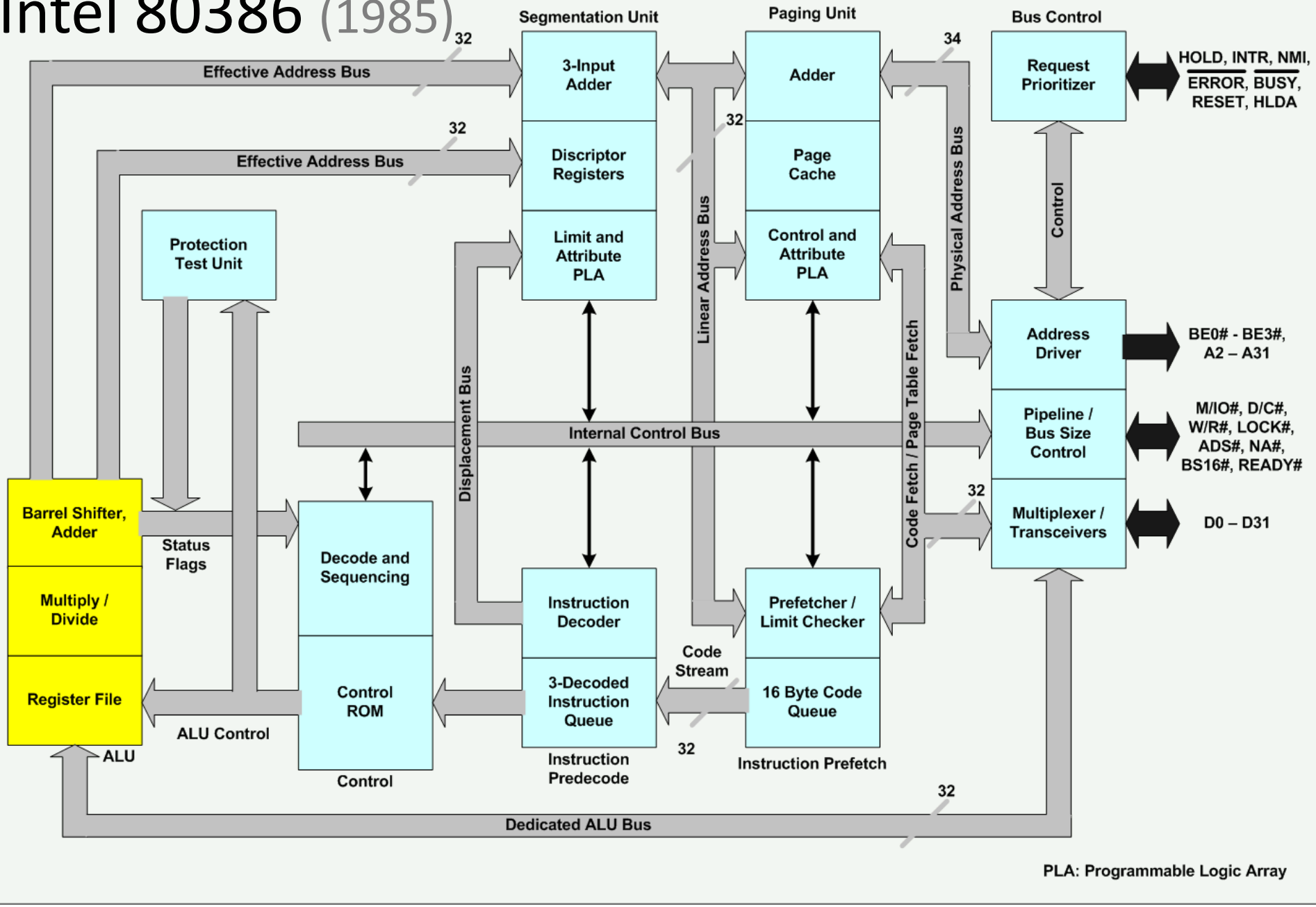
Intel 80186 (1982)



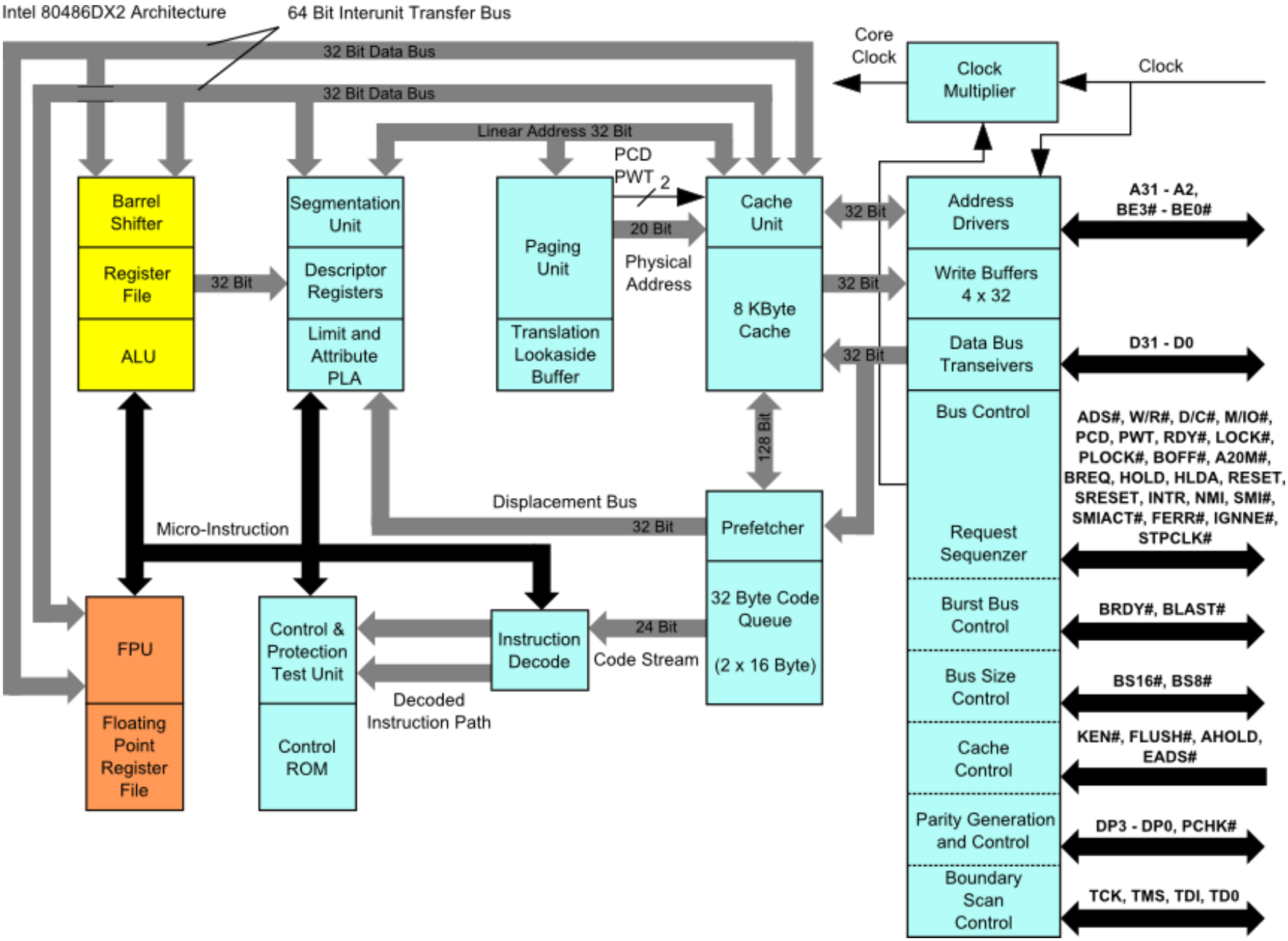
Intel 80286 (1982)



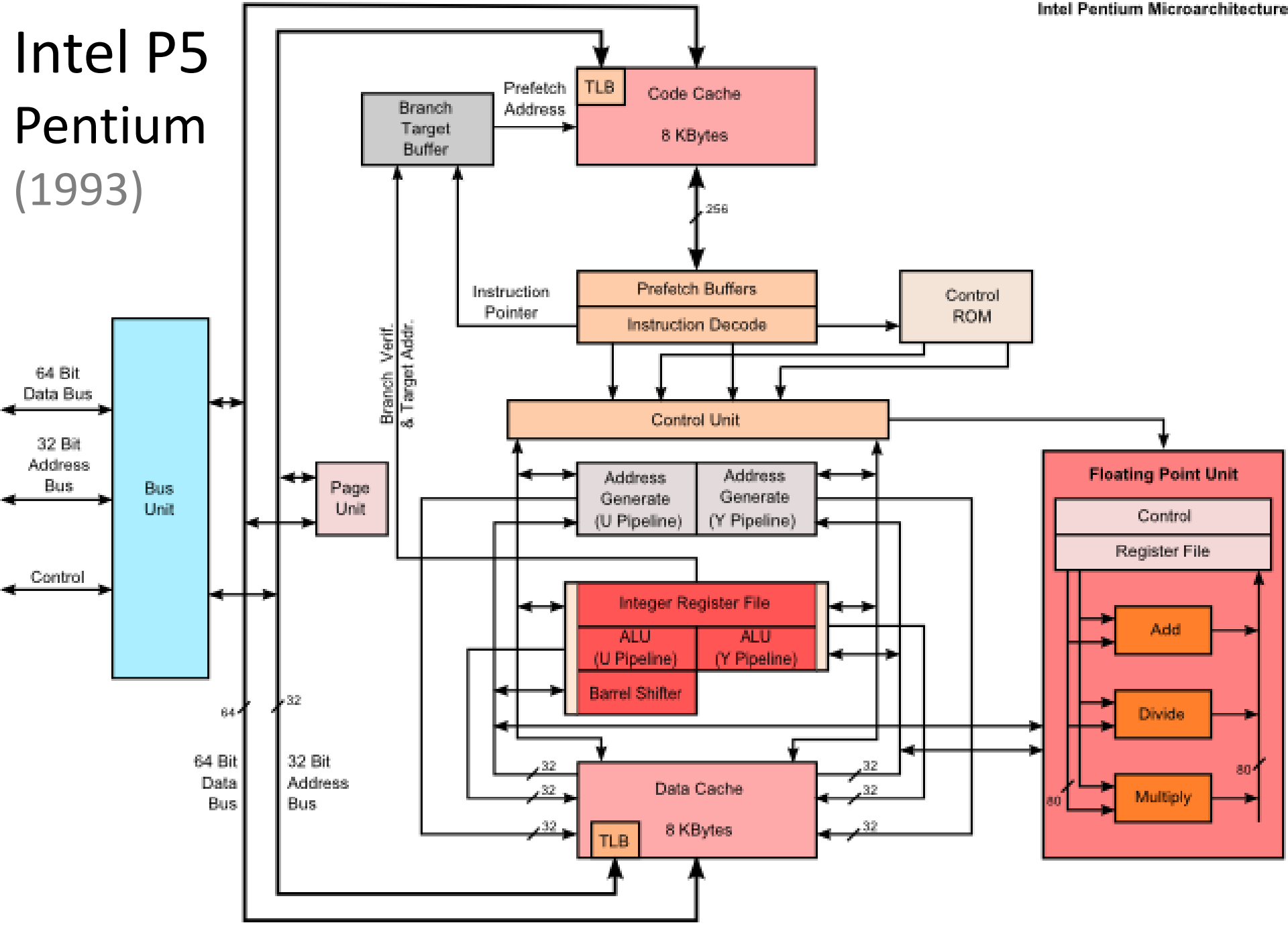
Intel 80386 (1985)



Intel 80486 (1989)



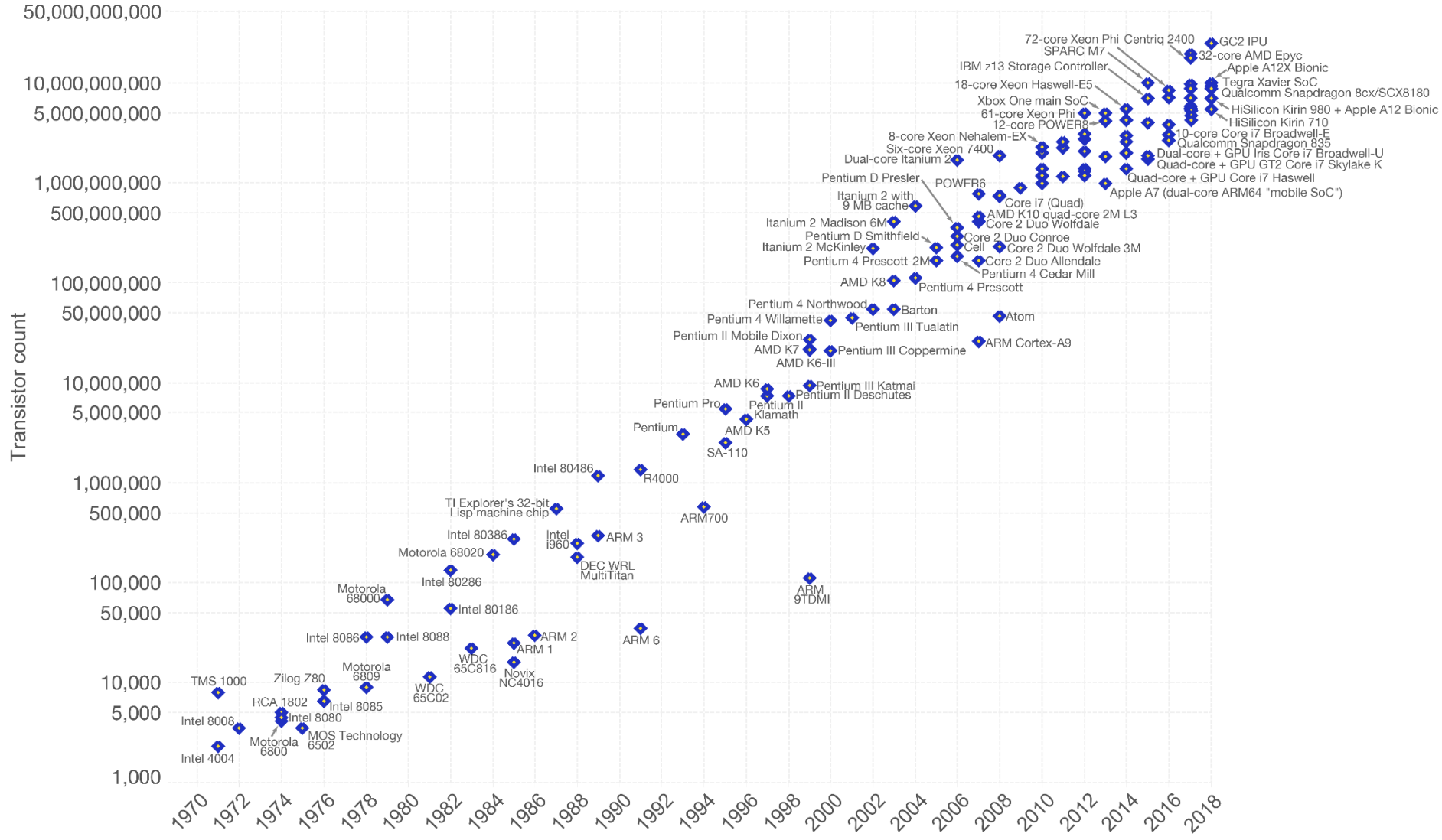
Intel P5
Pentium
(1993)



Moore’s Law – The number of transistors on integrated circuit chips (1971-2018)

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

Our World in Data



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

The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Arquitectura x86 es de las más utilizadas en la actualidad

- El conocimiento de esta arquitectura resulta fundamental para aplicaciones complejas.
- Presenta algunas diferencias claves con la arquitectura del computadores básico.
- Puede definirse como Von Neuman + CISC.
- Veremos la versión de 16 bits, usada hasta el 286 (386 es de 32 bits).
- Actualmente se utiliza la ISA x86-64, propuesta y popularizada por:

AMD

Veamos primero la microarquitectura

- 4 registros de propósito general de 16 bits (AX, BX, CX, DX), divisibles en sectores altos y bajos (AX = AH | AL) (¿Por qué?).
- BX se utiliza además para el direccionamiento indirecto (registro base).
- 2 registros de 16 bits para 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.

Veamos primero la 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, A.
- Direcciones de memoria de 16 bits.
- Palabras de memoria de 8 bits.
- Stack en memoria, SP apunta al último elemento ingresado al stack.

ISA x86 es más compleja que la del computador básico

- 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

Definición y uso de labels/variables es distinto en assembly x86 16 bits

- 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

¿Cómo queda la memoria luego de declarar variables?

- Existen 4 variables declaradas a partir de la dirección 0:
  - var1 db 0x0A y var2 dw 0x07D0
  - arr1 db 0x01, 0x02, 0x03 y arr2 dw 0x0A0B, 0x0C0D

Variable	Dirección (16 bits)	Palabra (8 bits)
var1	00	0x0A
var2	01	0xD0
	02	0x07
arr1	03	0x01
	04	0x02
	05	0x03
arr2	06	0x0B
	07	0x0A
	08	0x0D
	09	0x0C

Ejemplo Multiplicación: Código Java

```
public static void mult()
{
    int a = 10;
    int b = 200;
    int res = 0;
    while(a > 0)
    {
        res += b;
        a--;
    }
    System.out.println(res);
}
```

Ejemplo Multiplicación: Código Assembly x86

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

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
```

ISA x86 aprovecha ventajas de CISC

- Arquitectura CISC permite incluir instrucciones aritméticas complejas como (I)MUL y (I)DIV, que utilizan varios ciclos:

MUL *op* => AX = AL × *op*

MUL *op* => DX|AX = AX × *op*

DIV *op* => AL = AX ÷ *op* (resto en AH)

DIV *op* => AX = DX|AX ÷ *op* (resto en DX)

```
;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
```

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2343 – Arquitectura de Computadores

Arquitectura x86 – Elementos Básicos

Profesor: Hans Löbel