Daniel Pareja De La Flor

CUESTIÓN A

Estudio detallado que contraste las arquitecturas ARM/x86.

ASPECTOS BÁSICOS

- ARM y x86 son conjuntos de instrucciones

- x86 y ARM hacen referencia a los procesadores y a los lenguajes de programación

LENGUAJE DE PROGRAMACIÓN

- Son a bajo nivel. Programas no compatibles.

- A parte, ambos usan lenguaje ensamblador (un lenguaje de programación intermedio entre alto nivel como por ejemplo C++ y binario).

X86

- Arquitectura CISC
- Instrucciones complejas tamaño variable
- Ejecución registro-memoria
- Simplifica la estructura de programación
- Alto consumo energético
- Más espacio físico

ARM

- Arquitectura RISC
- Instrucciones simples tamaño fijo
- Sólo las instrucciones de carga y almacenamiento acceden a la memoria de datos (registro-registro)
- Facilita el paralelismo en la ejecución de instrucciones
- Bajo consumo energético
- Poco espacio

DIFERENCIAS

- x86: operaciones complejas con mayor velocidad

- ARM: operaciones sencillas poco consumo

QUÉ ES MEJOR PARA QUÉ

- x86: usado desde los 80. Operaciones complejas.

- ARM: para dispositivos móviles optimizar energia.

X86 EN LA ACTUALIDAD

- x86 de 32 bits compatible con 16 bits

- PCs actuales de 64 bits x86-64 compatible con 32 bits y con16 bits

ARM EN LA ACTUALIDAD

- ARM es cada vez más importante en el mercado
- ARM principalmente de 32 bits

OTROS DATOS

- Los chips x86 creados por Intel en 1978
- Los chips de ARM son propiedad de ARM Holdings
- ARM Holdings permite a otras empresas usar ARM por contra Intel es más restrictiva
- La única empresa que también usa x86 es AMD

Miguel Angel Fernández Montilla

CUESTIÓN B

Idear un programa (deberá ser una idea original –no una copia-) y codificarlo en ambas arquitecturas ARM/x86. Identificar pros y contras.

PROGRAMA ELEGIDO

- Pide por pantalla al usuario que introduzca un número y, dado ese número, mostrar si es perfecto o no

- Como aclaración, un número es perfecto si la suma de sus divisores dan dicho número

```
    AREA    ejer,CODE,READWRITE

SWI_Salir   EQU &11 ;salida del programa
SWI_write0  EQU &2  ;muestra por pantalla
SWI_ReadC   EQU &4  ;leer un caracter del teclado

        ENTRY

INICIO
    ADR r0, cad1    ;Obtenemos dirección de la cadena
    SWI SWI_write0  ;interrupción para mostrar cadena
    SWI SWI_ReadC   ;Punto de ruptura para introducir el
                    ;dato en R0

    MOV r1,r0       ;Movemos el valor introducido en r0
                    ;a r1

    SUB r1,r1,#48   ;Restamos 48 al valor ascii del
                    ;dato introducido para poder operar
                    ;con él

    MOV r2, #2      ;Para dividir el numero entre dos y
                    ;no comprobar con números superiores
                    ;a su mitad

    MOV r4, #0      ;Inicializamos el registro r4 a 0.
                    ;Y así poder usar la division

    MOV r3, #1      ;Inicializamos el registro r3 a 1,
                    ;que contendrá el cociente.

    CMP r1, #2      ;Comprobamos que el número
                    ;introducido no es menor de 2

    BLT NO_PERFECTO;Saltamos cuando el número
                    ;introducido sea menor de 2
    MOV r7, r1      ;Duplicamos el número introducido
```

ARM

- Todo programa en ARM comienza con la directiva AREA
- A continuación se asignan un valor numérico a esos nombres
- La directiva ENTRY declara el punto de entrada del programa
- Debajo de la etiqueta INICIO tenemos la inicialización de registros

ARM

```
                    ;Dividimos el número introducido
                    ;entre 2

MITAD   SUB  r7, r7, r2;restamos 2 a r7 (contiene el
                    ;numero introducido)

        ADD r3, r3, #1  ;vamos sumando 1 al divisor
                    ;que será el contador del bucle
                    ;siguiente

        CMP r7, r2      ;Comprobamos si podemos seguir
                    ;dividiendo

        BGE MITAD       ;salta mientras r7 sea mayor o
                    ;igual que 2.

        MOV r8, #2      ;r8 contendrá los candidatos a
                    ;divisor

        MOV r7, r1      ;Volvemos a introducir el número
                    ;leído en r7

        MOV r6, #1      ;r6 Acumulará la suma de los
                    ;divisores del número introducido
```

- División en ARM que se debe realizar con un bucle. Ya que no hay instrucción que permita realizarla de forma sencilla

- En este trozo de código dividimos el número a la mitad, para no comprobar con números superiores a si mitad y buscar sus divisores

ARM

```
COMPROBAR
                    ;Para comprobar si los candidatos
                    ;a divisor lo son realmente

    SUB  r7, r7, r8     ;Restamos el candidato a divisor
                        ;al número introducido

    ADD r4, r4, #1      ;r4 irá almacenando el cociente
    CMP  r7, r8         ;Comparamos dividendo con divisor
    BGE COMPROBAR       ;Salta mientras el dividendo no
                        ;sea menor que el divisor

    CMP  r7, #0         ;Comprueba si el resto es cero
    BNE SIGUIENTE       ;Si el resto es distinto de 0,
                        ;salta a SIGUIENTE

    ADD  r6, r6, r8     ;Si el resto es 0, sumamos al
                        ;registro r6 el divisor encontrado
```

- Una vez que tengamos un candidato a divisor comprobamos si realmente es divisor
- Si no lo es, el número saltará a SIGUIENTE
- Si es divisor lo sumamos al registro r6

ARM

```
SIGUIENTE   CMP r8, r3       ;Comprobamos si ya hemos probado
                             ;con todos los posibles divisores

            BEQ RESULTADO    ;Salta cuando hayamos realizado
                             ;todas las iteraciones previstas.

            ADD  r8, r8, #1  ;Incrementamos en 1 para tener un
                             ;nuevo candidato a divisor.

            MOV r7, r1       ;Reiniciamos el dividendo
            B COMPROBAR      ;Volvemos para comprobar el
                             ;siguiente candidato a divisor

RESULTADO   CMP r1, r6       ;Comprobamos si la suma de los
                             ;divisores es igual al número
                             ;introducido

            BNE NO_PERFECTO ;Si son distintos saltamos a
                             ;NO_PERFECTO

            ADR  r0, cad2    ;Obtenemos dirección de la cadena
            SWI  SWI_write0 ;interrupción para mostrar cadena
            B FIN
```

- Aquí se comprueba que el número que nos llegó del apartado anterior era el último o no
- Si es así, salta a RESULTADO
- Si no es el último, se incrementa el número y lo volvemos a comprobar

- En RESULTADO se comprueba la suma divisores con el número introducido
- Si son iguales mostramos la cadena2
- Si son distintos salta a NO_PERFECTO

ARM

```
NO_PERFECTO     ADR r0, cad3    ;Obtenemos dirección de la
                                ;cadena

                SWI SWI_write0  ;interrupción para
                                ;mostrar cadena

FIN             ADR  r0, cad4   ;Para volver a ejecutar el
                                ;programa

                SWI  SWI_write0;Interrupción para mostrar
                                ;cadena

                SWI  SWI_ReadC  ;Punto de ruptura para
                                ;introducir el dato en R0

                CMP r0, #49     ;Comprobamos si desea
                                ;volver a ejecutar el
                                ;programa

                BEQ INICIO      ;Si el usurario ha
                                ;introducido un 1 repetimos
                                ;el programa


                SWI SWI_Salir   ;Interrupción para salir
```

- En NO_PERFECTO se muestra la cadena3
- Luego tenemos el final del programa

```
cad1 =  "Introduzca un número entre 1 y 9", &0a, &0d, 0
cad2 =  "El numero introducido es perfecto", &0a, &0d, 0
cad3 = "El numero introducido no es perfecto", &0a, &0d, 0
cad4 =  "Para volver a ejecutar el programa introduzca 1", &0a, &0d, 0

    END
```

ARM

- Por último tenemos lo que mostrará cada cadena.
- Y, lo que no puede faltar, es la directiva END que muestra el final del fichero.

X86

```
.Const

.Data
NUM DQ 0


.Code

start:
;Inicio del programa principal.

invoke puts, "----- Programa para comprobar si un numero es perfecto -----"
invoke puts, " "
invoke printf, "----- Introduzca un numero entero positivo :  "
invoke scanf, "%d", Addr NUM
invoke puts, " "
mov ebx, 2      ;Para dividir el número entre dos y así no comprobar
                    ;números superiores a su mitad
mov eax, [NUM]  ;Introducimos el número en eax
mov edx, 0      ;Para poder usar DIV sin problemas
div ebx         ;Lo dividimos entre dos mov ecx, eax
                ;Almacenamos el cociente en ecx para que sirva de contador
                ;para LOOP
mov edi, 1      ;Contendrá los candidatos a divisores
mov esi, 0      ;Para ir sumando los divisores
```

- A diferencia del ARM, aquí comenzamos con 3 secciones .CONST .DATA .CODE
- .CONST: declaración de constantes
- .DATA: declaración de variables
- .CODE: desarrollo del programa
- En esta parte de código vemos la inicialización de los registros
- La división del número introducido entre dos para no operar con números superiores a su mitad

X86

```
comprobar:
 mov eax, [NUM] ;Introducimos el numero en eax para DIV
 div edi   ;Dividimos el número entre el candidato a
divisor
 cmp edx, 0  ;Comprobamos si el resto es cero
 jne > siguiente ;Salta si el resto no es cero
 add esi, edi ;Sumamos el divisor encontrado
```

- Dividimos nuestro número entre el candidato a divisor inicializado anteriormente.
- Si el resto es 0 suma el divisor encontrado.
- Si no lo es salta a SIGUIENTE

X86

```
siguiente:
 inc edi   ;Incrementamos en uno el candidato a divisor
 mov edx, 0  ;Eliminamos el posible resto almacenado para
poder utilizar DIV
 loop comprobar ;Iteramos mientras no se comprueben todos
los números entre 1 y NUM/2.

 cmp [NUM], esi  ;Comprobamos si la suma de los divisores
es igual al número introducido
 jne > no_perfecto ;Saltamos si no son iguales
 invoke puts, "----- El numero introducido es perfecto "
 jmp > fin
no_perfecto:
 invoke puts, "----- El numero introducido no es perfecto “


fin:
 invoke puts, " "
 Invoke system, "pause"
 Xor Eax, Eax
 Invoke ExitProcess, Eax

;Fin del programa principal

;Subprogramas:
```

- Incrementamos el candidato a divisor y lo devolvemos al apartado anterior hasta que no se comprueben todos los posibles divisores.
- Comprobamos si la suma de los divisores es igual al número introducido.
- Si son iguales, se muestra por pantalla que es perfecto.
- Si son distintos saltamos a NO_PERFECTO, que muestra que no es perfecto.
- Fin del programa

PROS Y CONTRAS

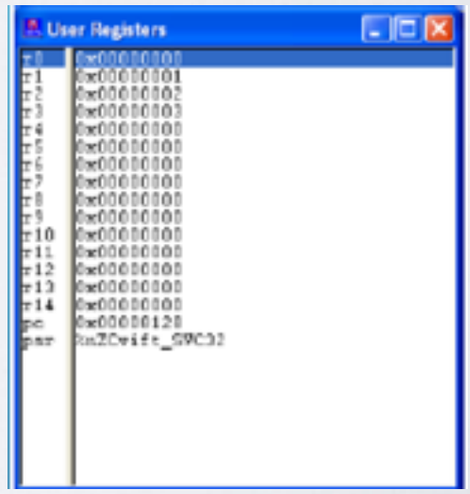
- En lenguaje ARM tenemos que realizar la división con un bucle. En x86, en cambio, utilizamos una sola instrucción, lo que hace que el código sea menos extenso.

- En ARM metemos el dato directamente en el registro. En x86 el dato lo lee gracias a la instrucción "Invoke scanf" y después de esto podemos asignárselo a un registro. Proporciona mayor comodidad, así podemos despreocuparnos de donde introducir el dato, como pasa en ARM.

- Una ventaja que tiene x86 es poder utilizar "invoke" y permitirnos llamar a algunas funciones utilizadas en C.

ARM

```
SUB r7, r7, r2
ADD r3, r3, #1
CMP r7, r2
BGE MITAD
```



PROS Y CONTRAS

- En lenguaje ARM tenemos que realizar la división con un bucle. En x86, en cambio, utilizamos una sola instrucción, lo que hace que el código sea menos extenso.

- En ARM metemos el dato directamente en el registro. En x86 el dato lo lee gracias a la instrucción "Invoke scanf" y después de esto podemos asignárselo a un registro. Proporciona mayor comodidad, así podemos despreocuparnos de donde introducir el dato, como pasa en ARM.

- Una ventaja que tiene x86 es poder utilizar "invoke" y permitirnos llamar a algunas funciones utilizadas en C.

X86

```
div ebx
```

```
Invoke scanf, "%d",Addr NUM
Invoke puts, " "
mov ebx, 2
mov eax, [NUM]
```

PROS Y CONTRAS

- En este caso concreto resulta mas cómodo utilizar "printf" y "scanf" para mostrar por pantalla y leer el dato en x86.
- En ARM debemos declarar la cadena y obtener su dirección para que así la muestre por pantalla.
- Esto último supone una ventaja a la hora de utilizar lenguaje x86 ya que estamos más acostumbrados a programar en C.

ARM

```
ADR r0, cad1     ;Obtenemos la direccion de la cadena
SWI SWI_write0   ;interrupcion para mostrar cadena
SWI SWI_ReadC    ;Punto de ruptura para introducir el dato en R0

cad1 = "Introduzca un número entre 1 y 9", &0a, &0d, 0
```

X86

```
invoke printf, "Introduzca un numero entero positivo : "
invoke scanf, "%d", Addr NUM
```

PROS Y CONTRAS

- La pila de x86 almacena valores temporales

- No la hemos utilizado para asemejar el código a ARM.

Juan Manuel Aspera Delgado
María Denís Rodríguez Fernández

CUESTIÓN C

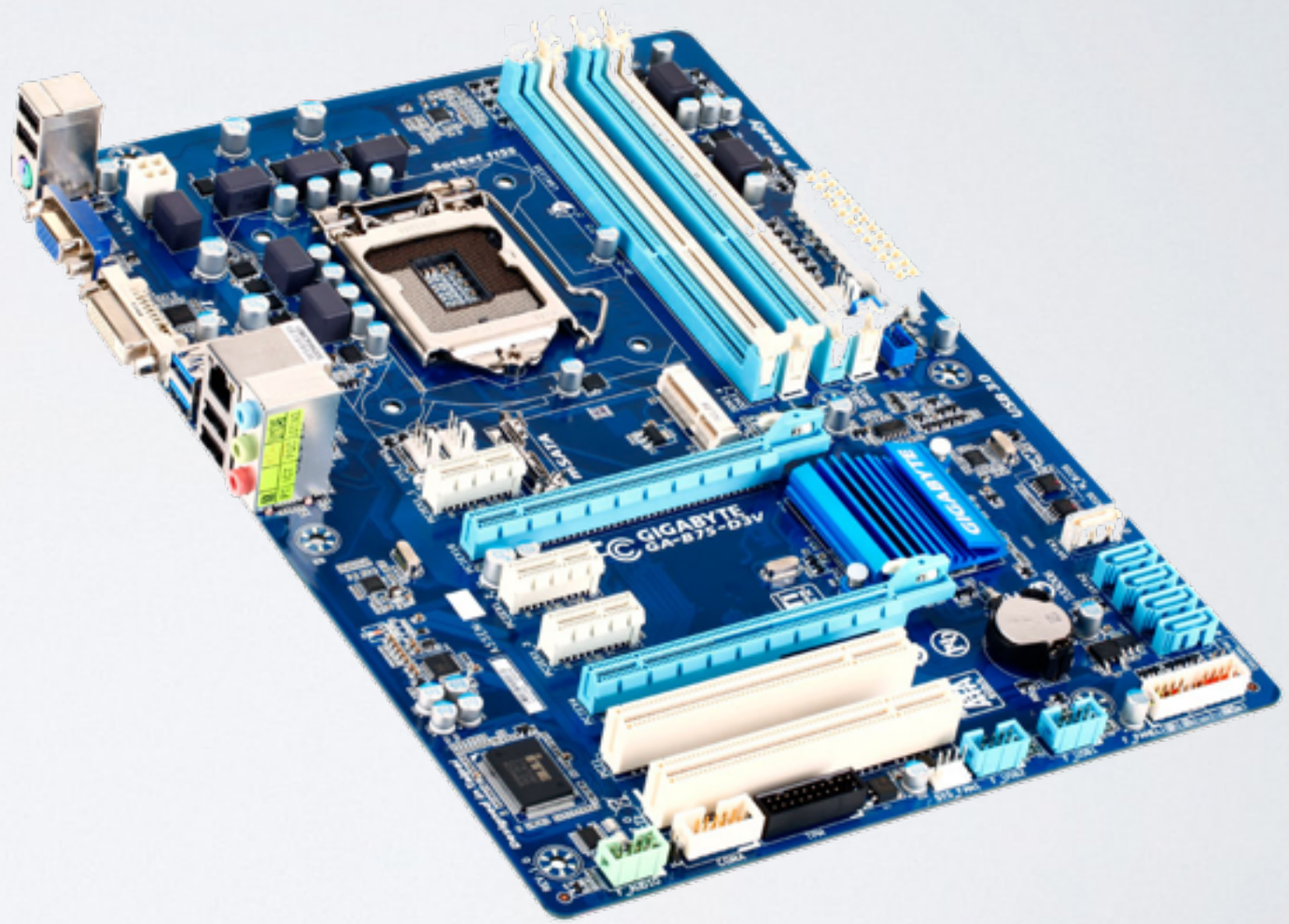
Dispones de 800 € ¿Qué ordenador de sobremesa te comprarías?. En este apartado el grupo deberá seleccionar los componentes del PC (sólo torre y elementos interiores); explicar en detalle cada uno de los componentes (deberá incluir imágenes ilustrativas) y, en especial, explicar los elementos de la placa base; presentar presupuesto.

PLACA BASE

Componentes

- Conectores Alimentación
- Chipset y Sockets
- BIOS y CMOS
- Ranuras Expansión
- Conectores E/S Internos
- Conectores E/S Externos

Gigabyte GA-B75-D3V

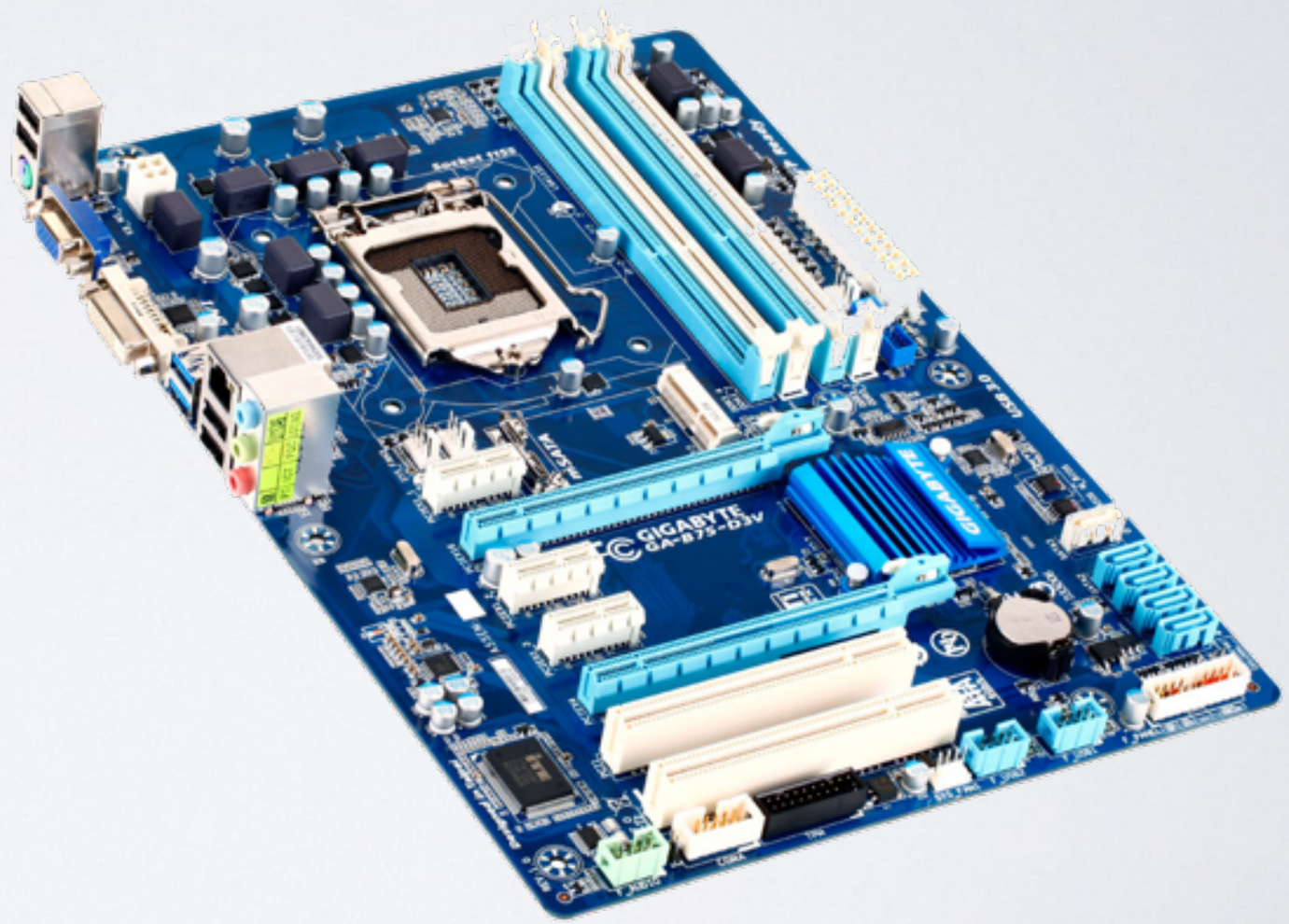


PLACA BASE

Componentes

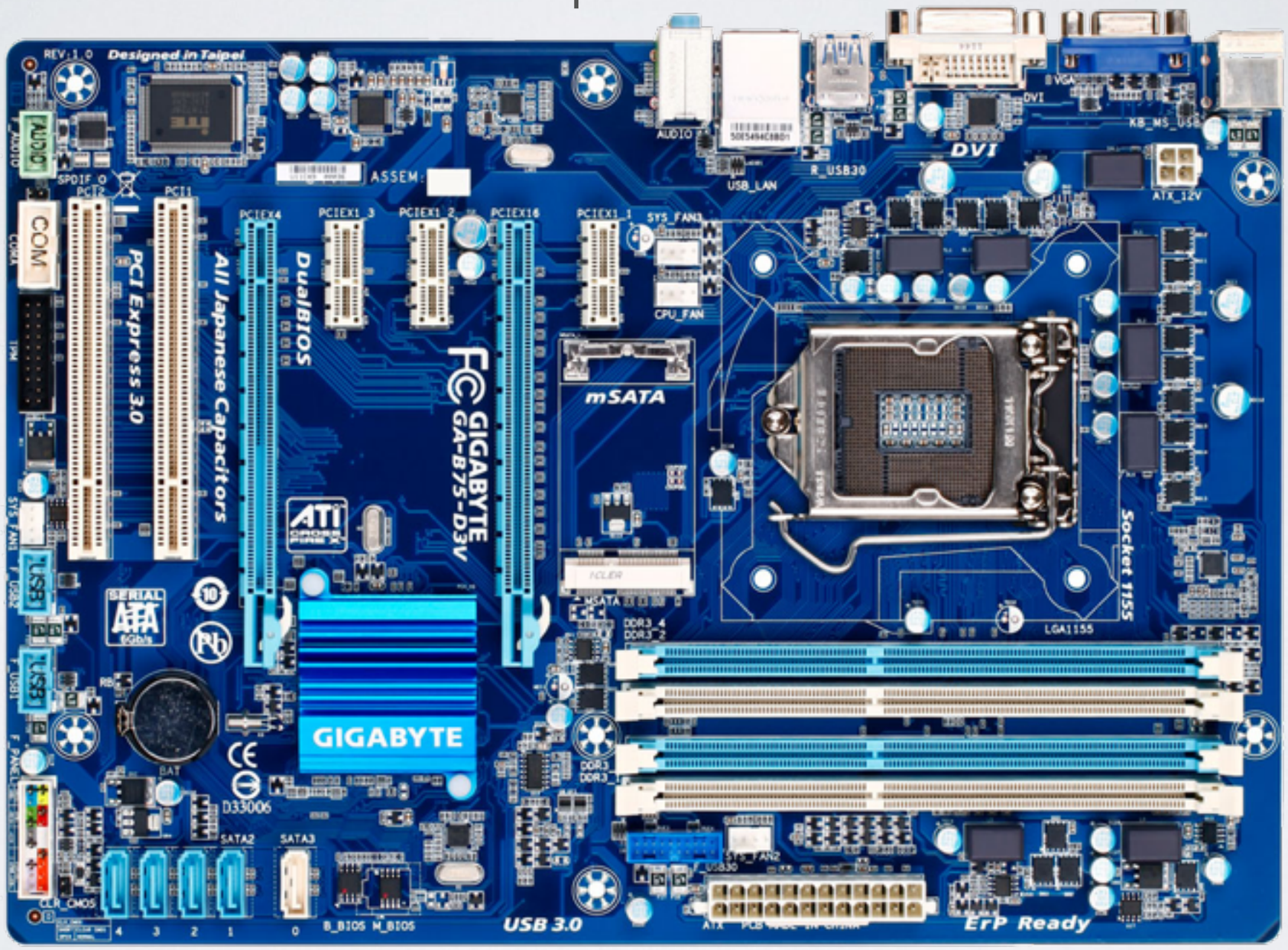
- Conectores Alimentación
- Chipset y Sockets
- BIOS y CMOS
- Ranuras Expansión
- Conectores E/S Internos
- Conectores E/S Externos

Gigabyte GA-B75-D3V



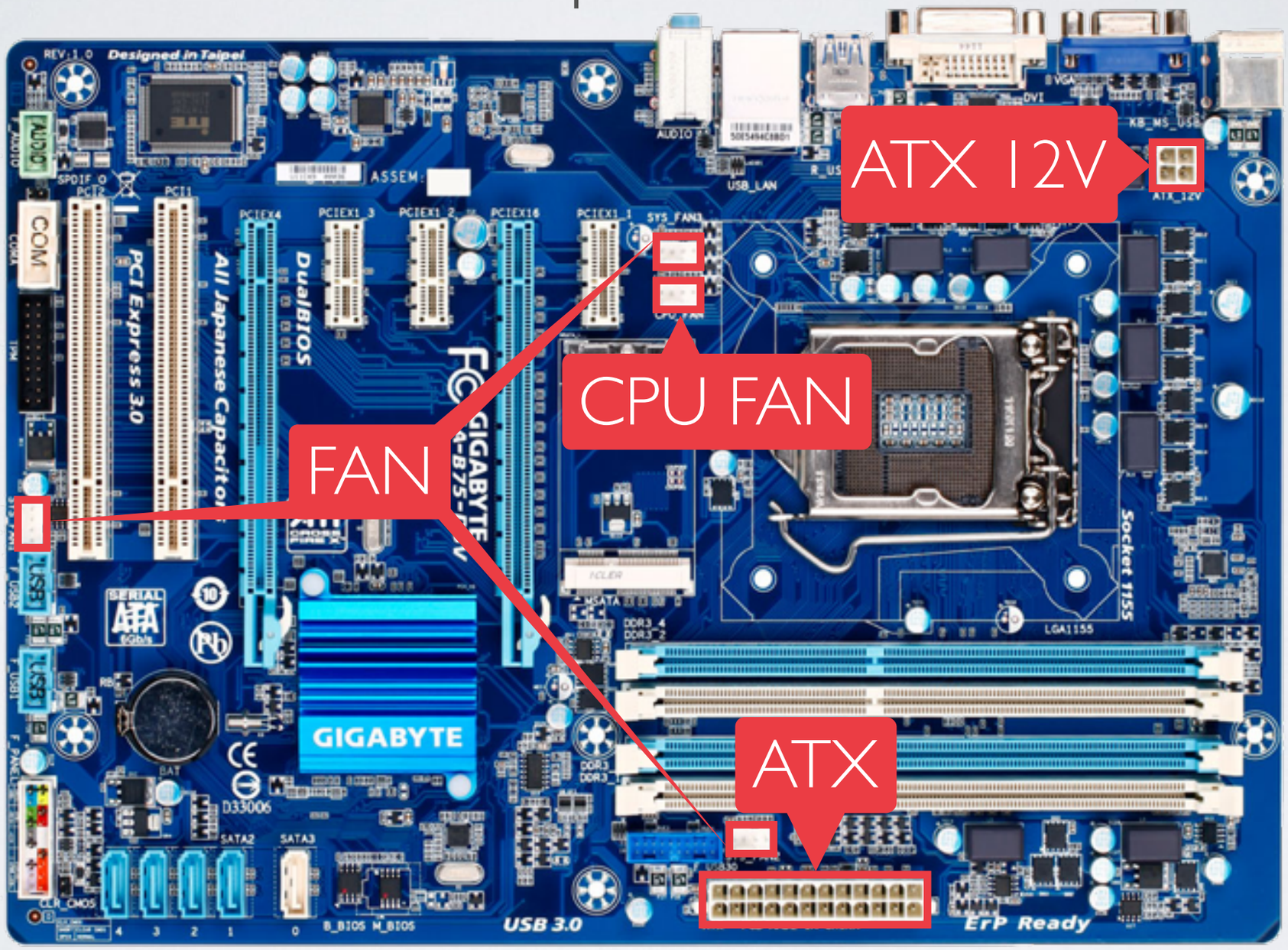
PLACA BASE

Componentes



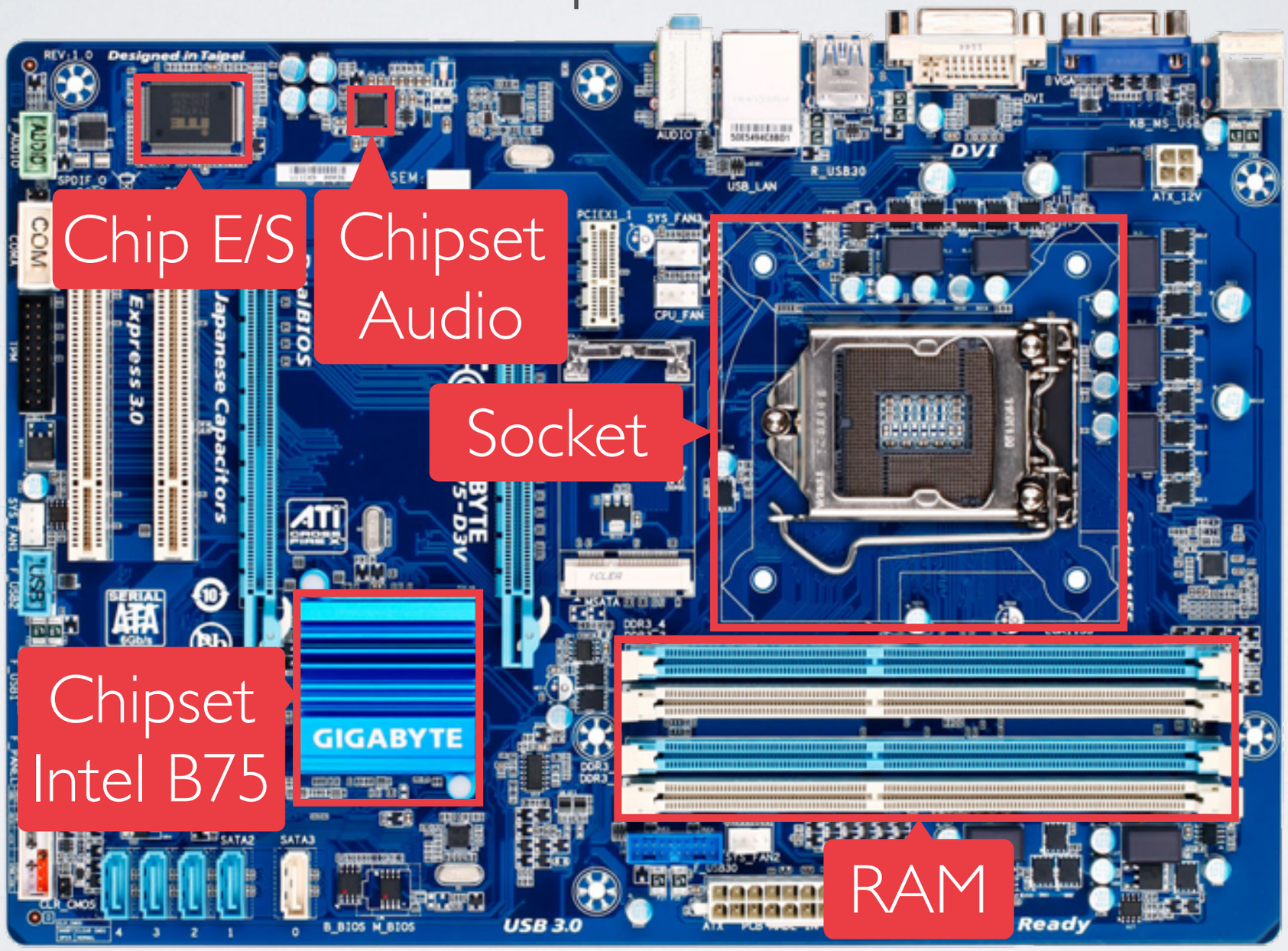
PLACA BASE

Componentes

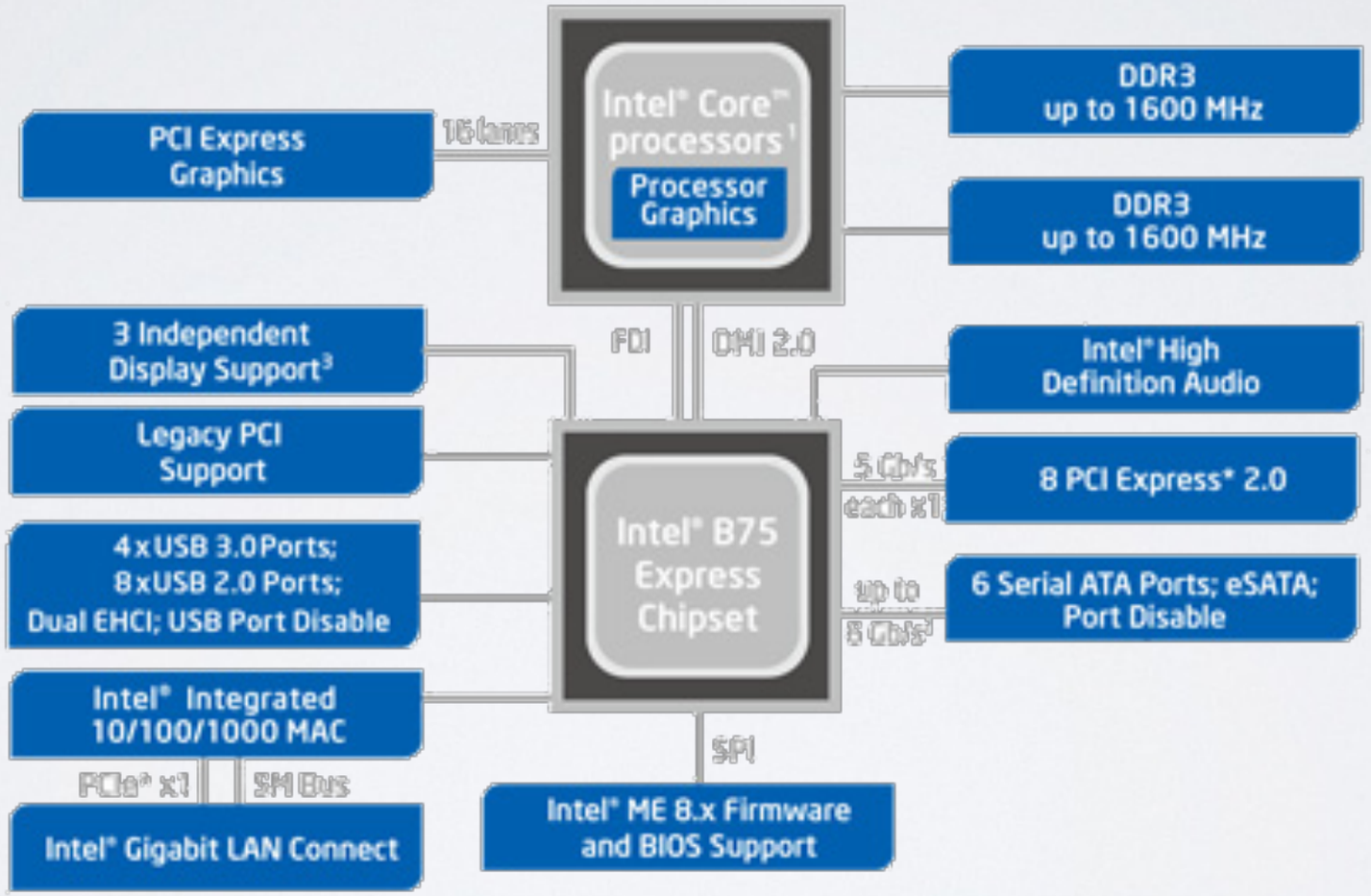


PLACA BASE

Componentes

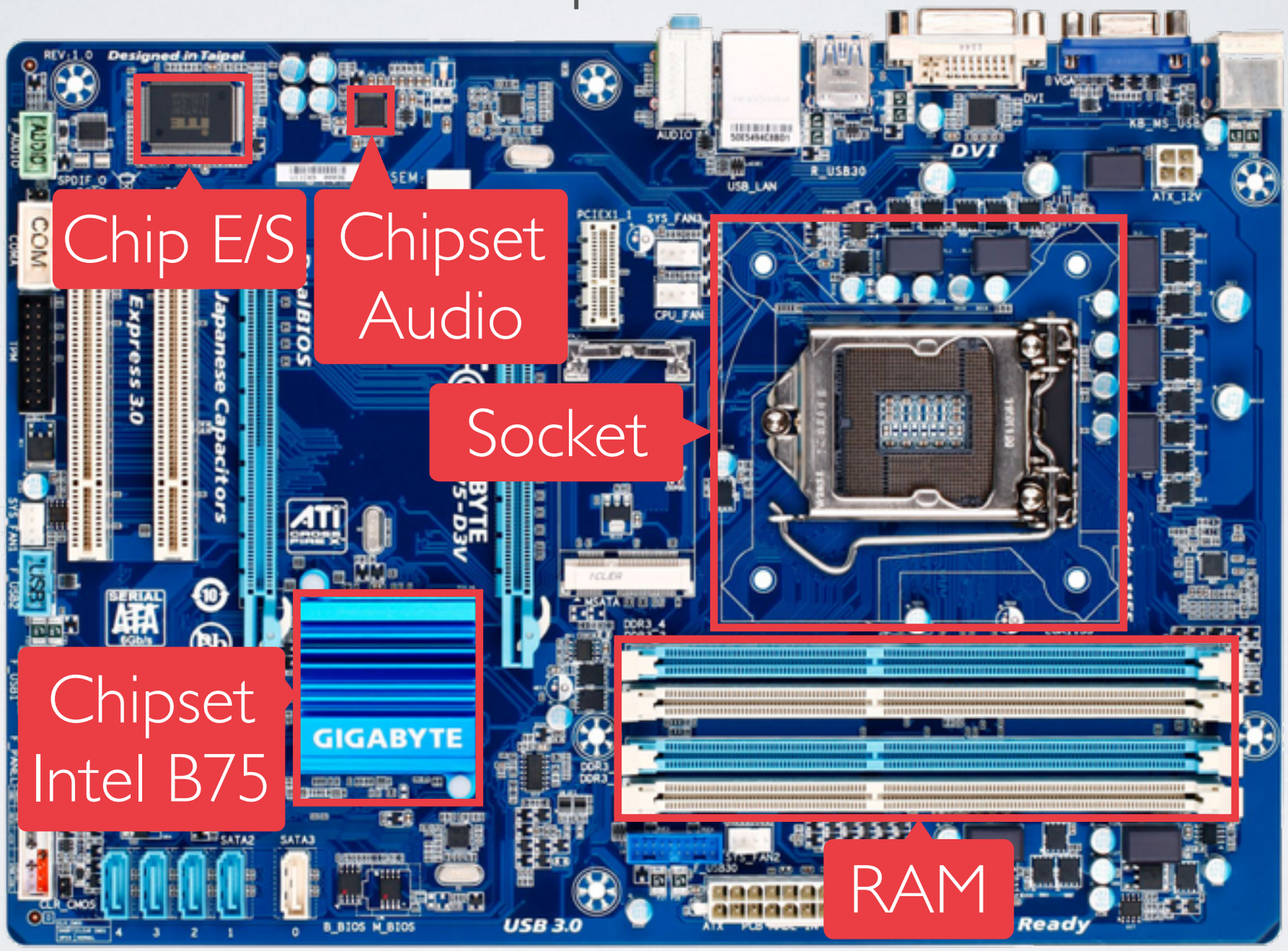


CHIPSET INTEL B75



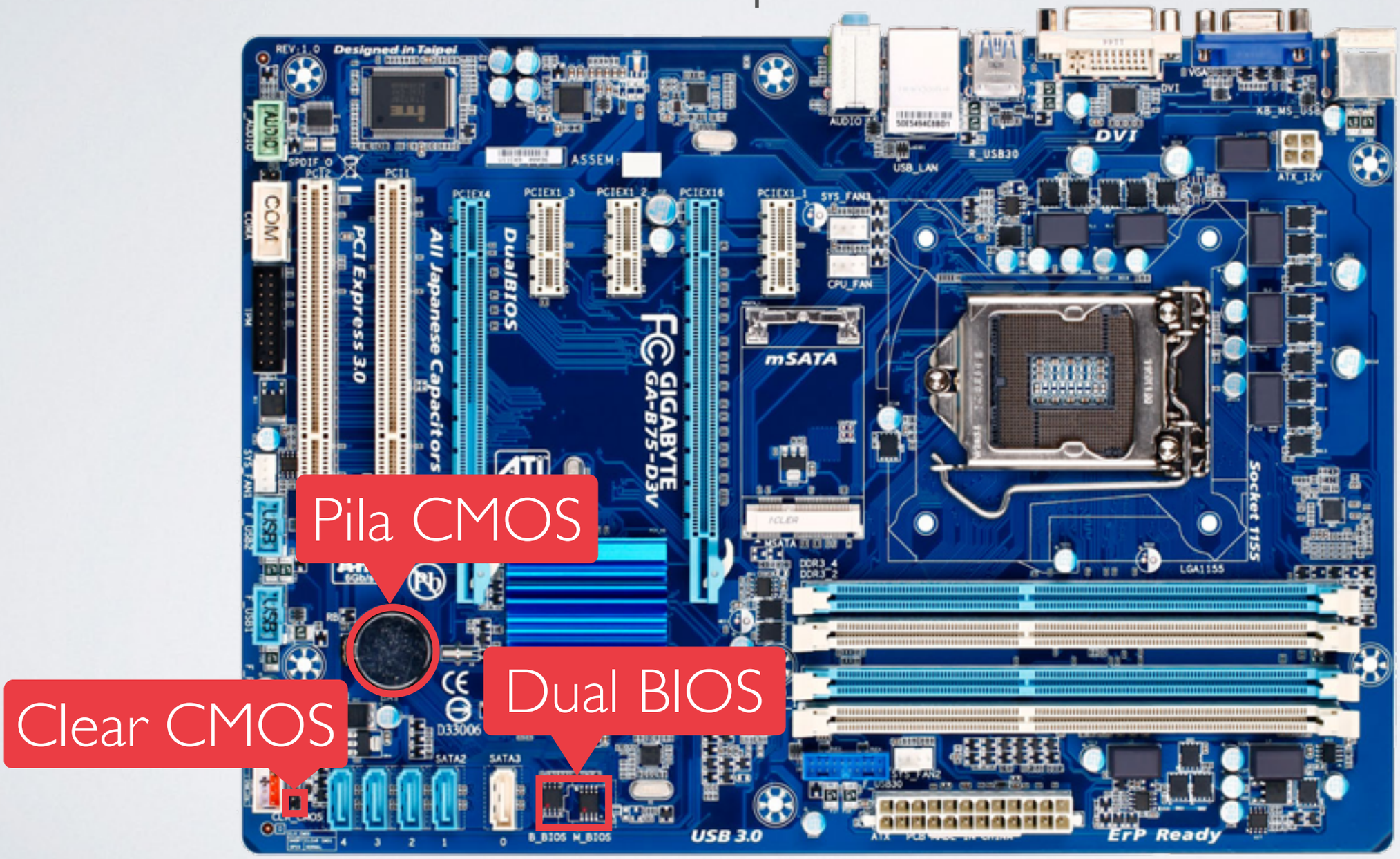
PLACA BASE

Componentes



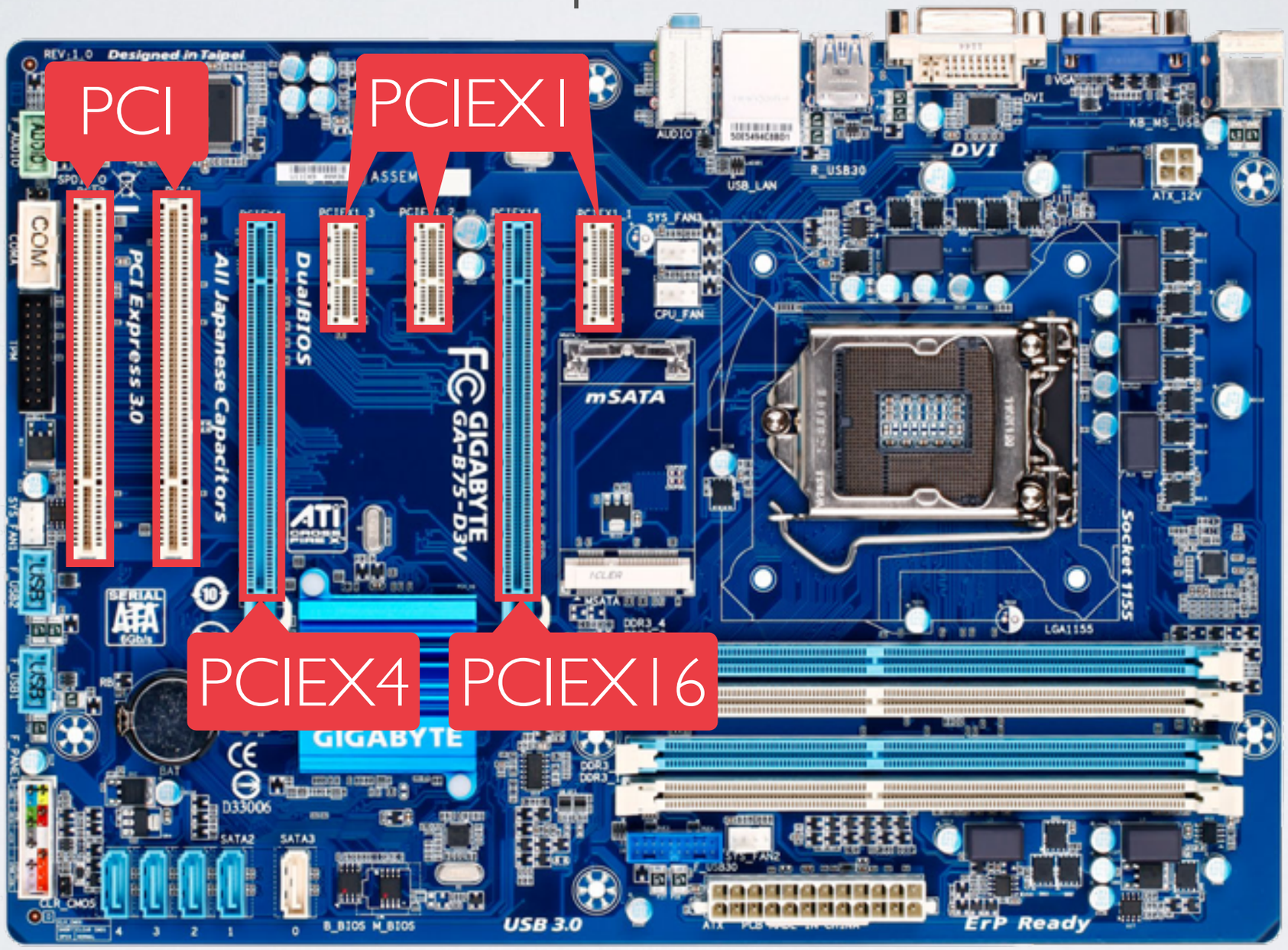
PLACA BASE

Componentes



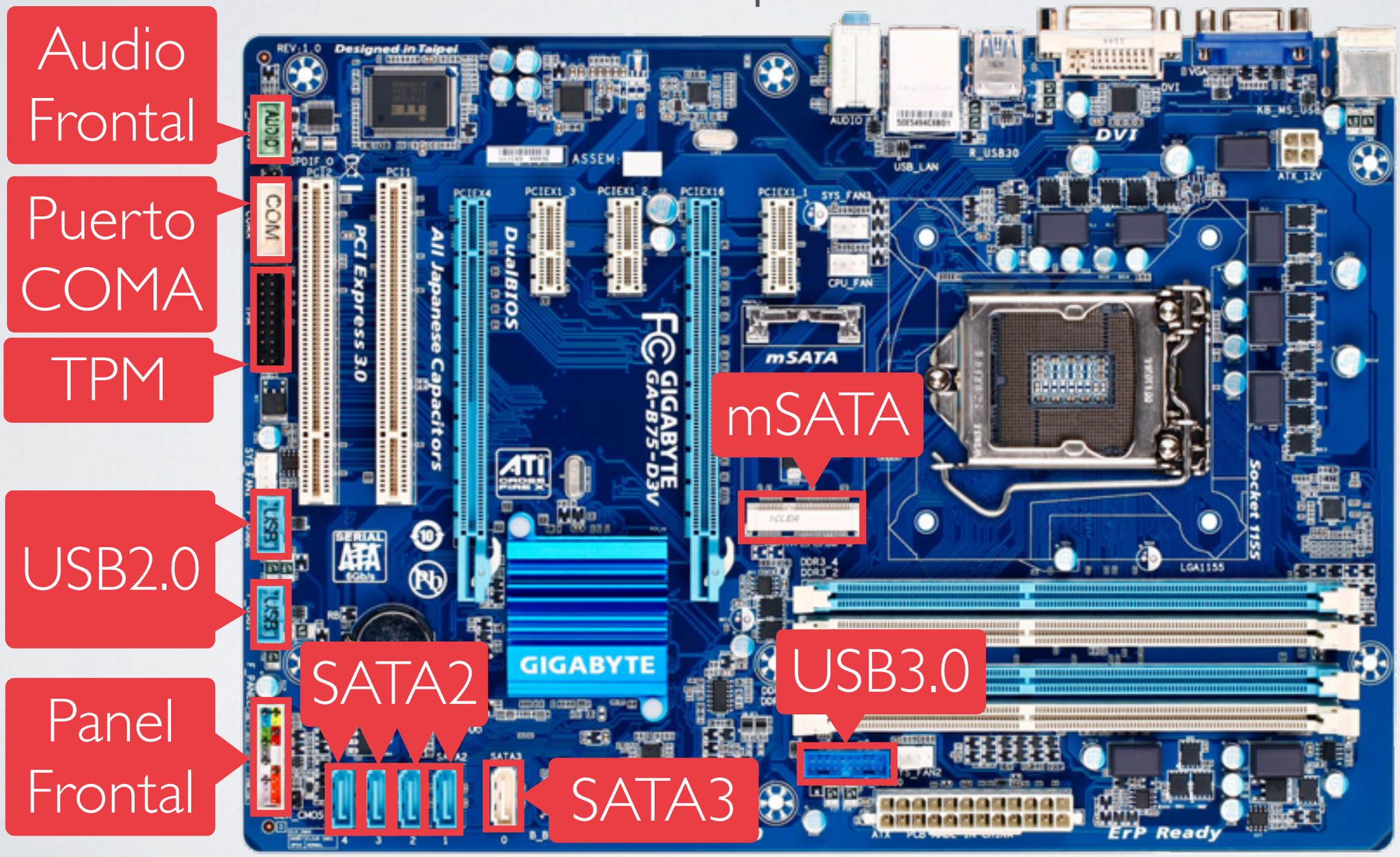
PLACA BASE

Componentes



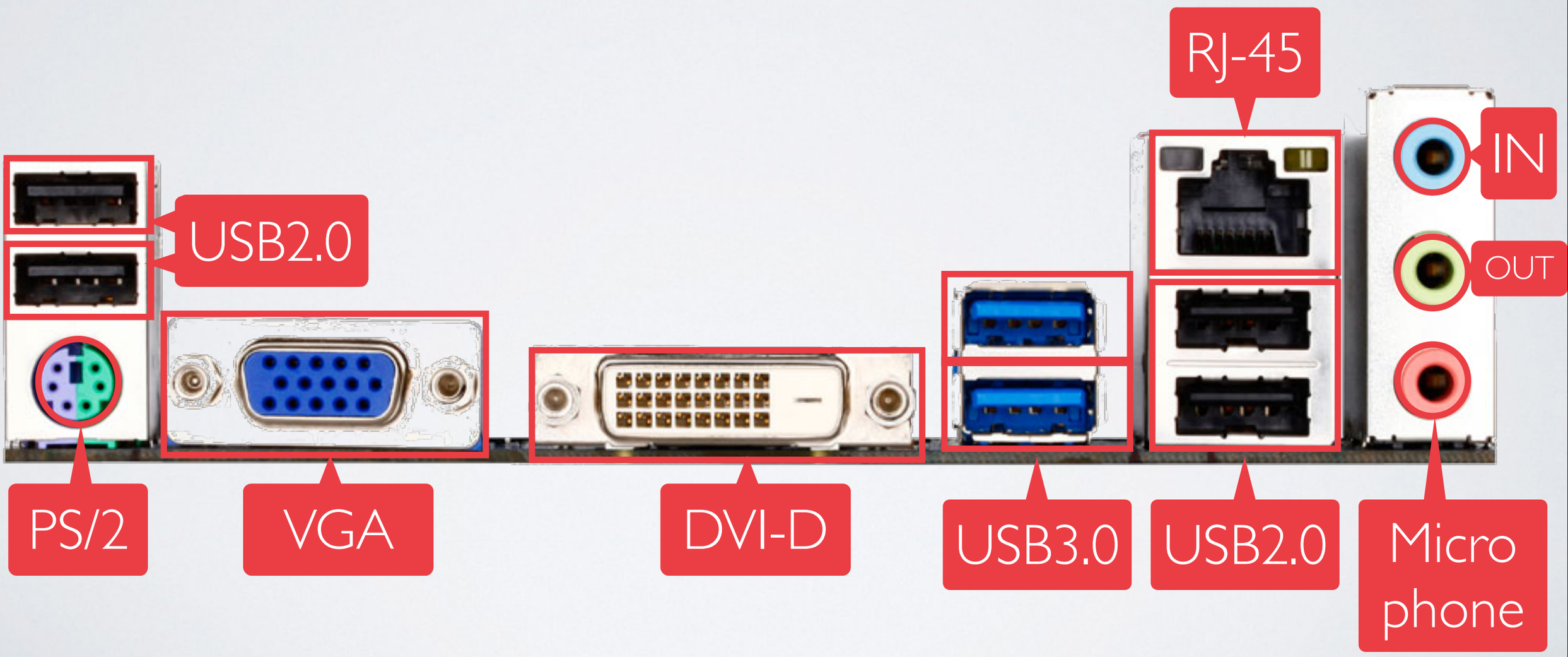
PLACA BASE

Componentes



PLACA BASE

Componentes



COMPONENTES

- Placa Base Gigabyte GA-B75-D3V
- Procesador
- Memoria Principal
- Tarjeta Gráfica
- Almacenamiento
- Lector Grabador DVD/CD
- Tarjeta Wifi
- Fuente de Alimentación
- Torre

PROCESADOR

Requisitos

- Actual para combatir la obsolescencia
- Potencia de sobra
- Capacidad de procesamiento paralelo
- Bajo consumo energético
- Socket LGA1155

PROCESADOR

Ofrece

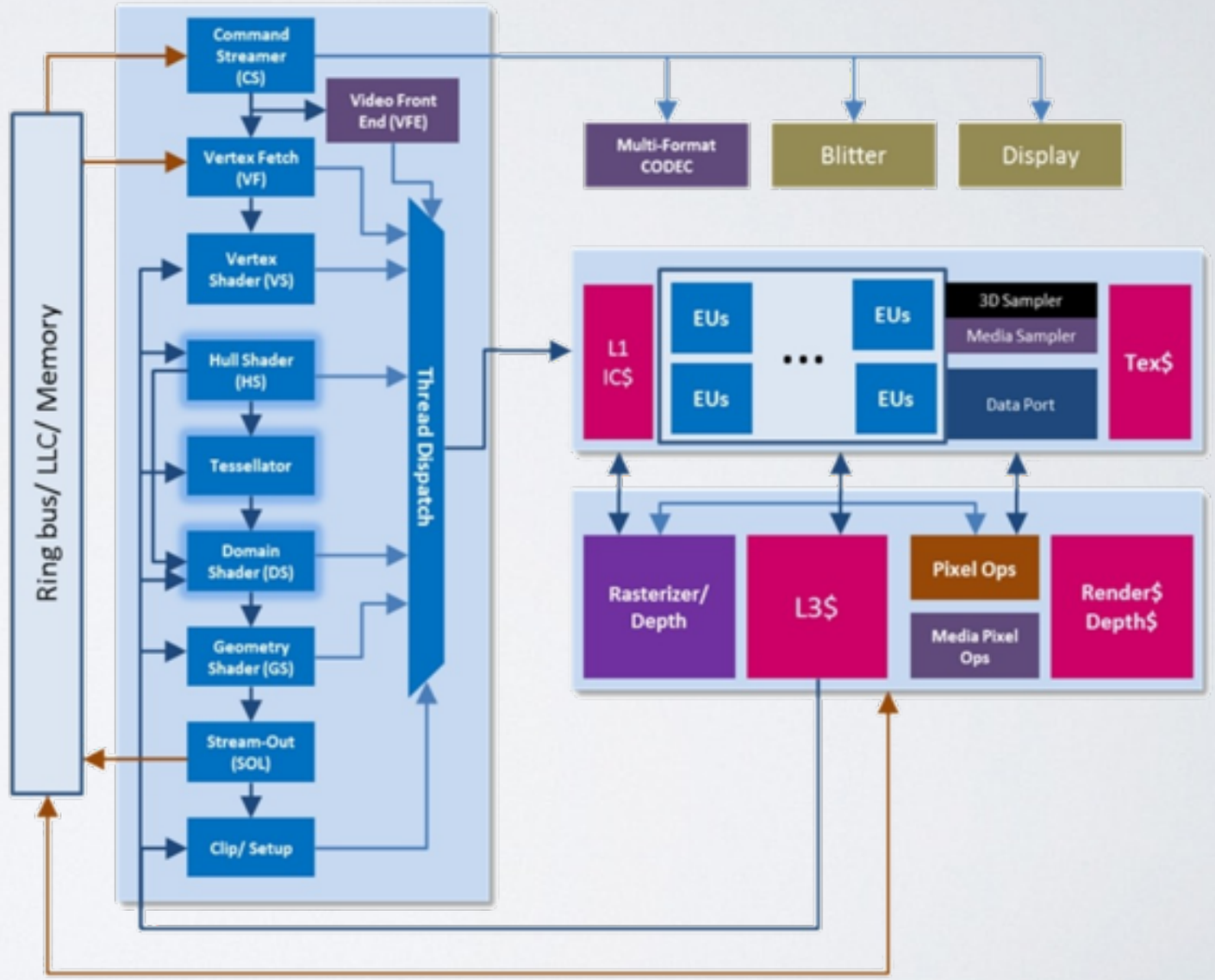
- 4 núcleos
- 2.8GHz Max. 3,3GHz
- Caché: L2 4x 256 kB y L3 6144 kB
- Ivy Bridge 22nm Q3 2013
- 65 W
- x86, MMX, SSE, SSE2, SSE3, SSSE3, x86-64, SSE4.1, SSE4.2, AES, AVX

Intel Core i5 3340S

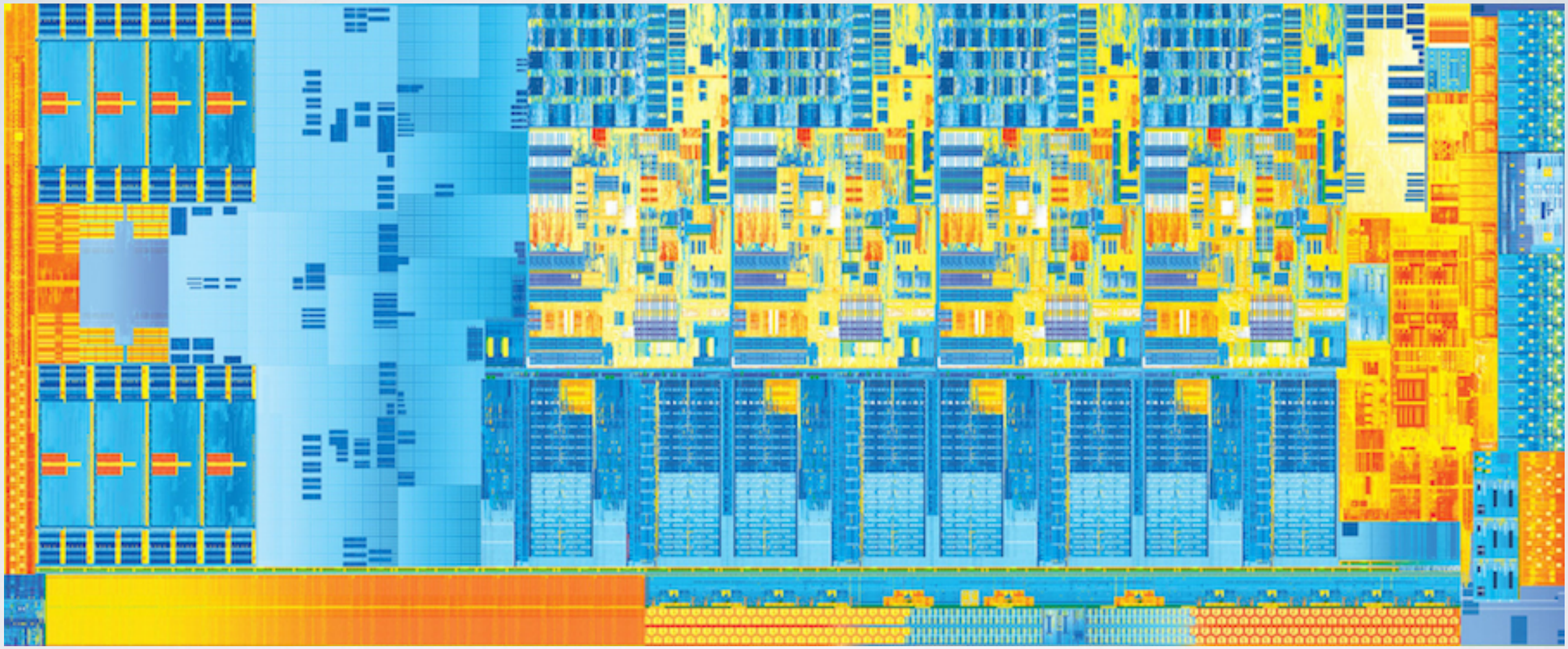
PROCESADOR

Además

- Tarjeta de Video Integrada
- Intel HD Graphics 2500 1350MHz
- DirectX 11.0, Shader 5.0

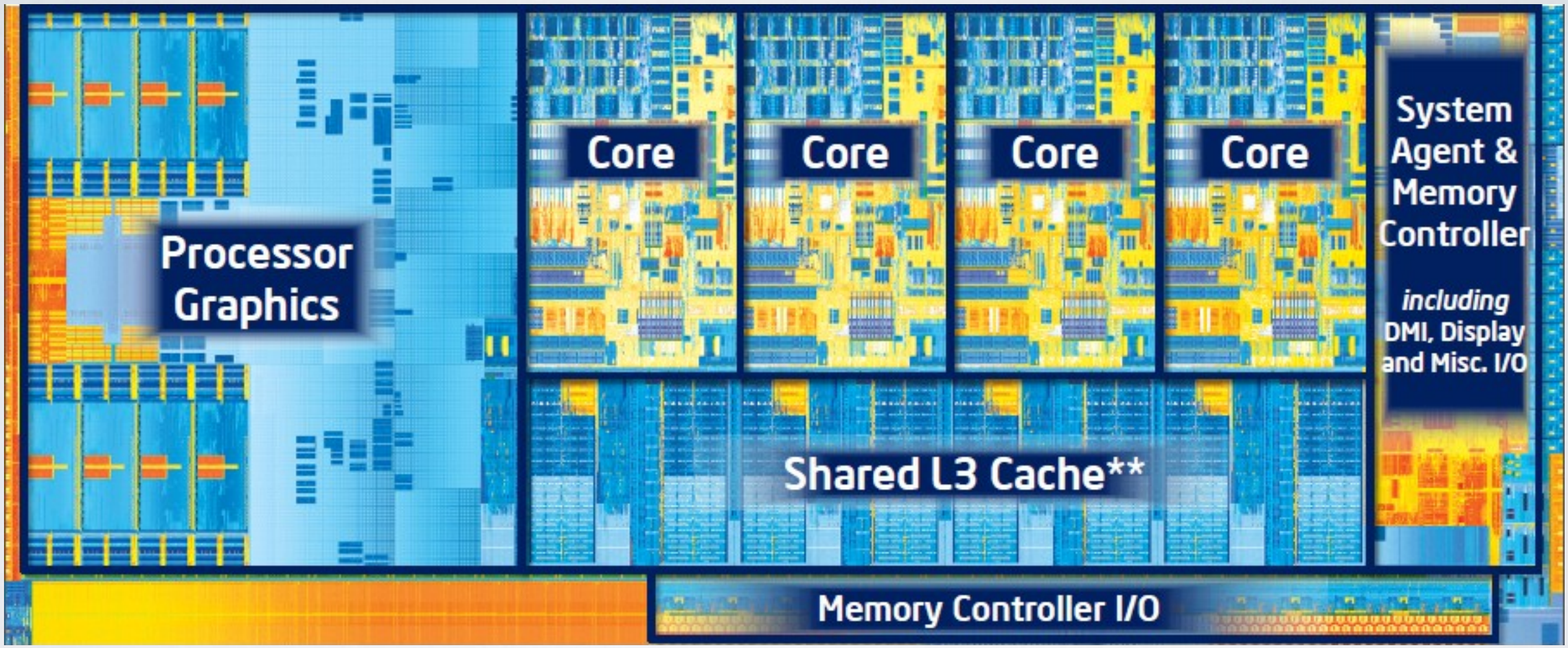


PROCESADOR



Intel Core Ivy Bridge

PROCESADOR



Intel Core Ivy Bridge

DISIPADOR DE CALOR CPU

Requisitos

- Socket LGA1155

DISIPADOR DE CALOR CPU

Ofrece

- Socket LGA1155

- Ruido 10-21 dB

Enermax ETS-T40-TB

MEMORIA PRINCIPAL

Requisitos

- 8GB
- Máxima eficiencia
- DDR3 1600 MHz
- Dual Channel

MEMORIA PRINCIPAL

Ofrece

Kingston Technology HyperX

- Modulos Gemelos Dual Channel
- 2x4GB

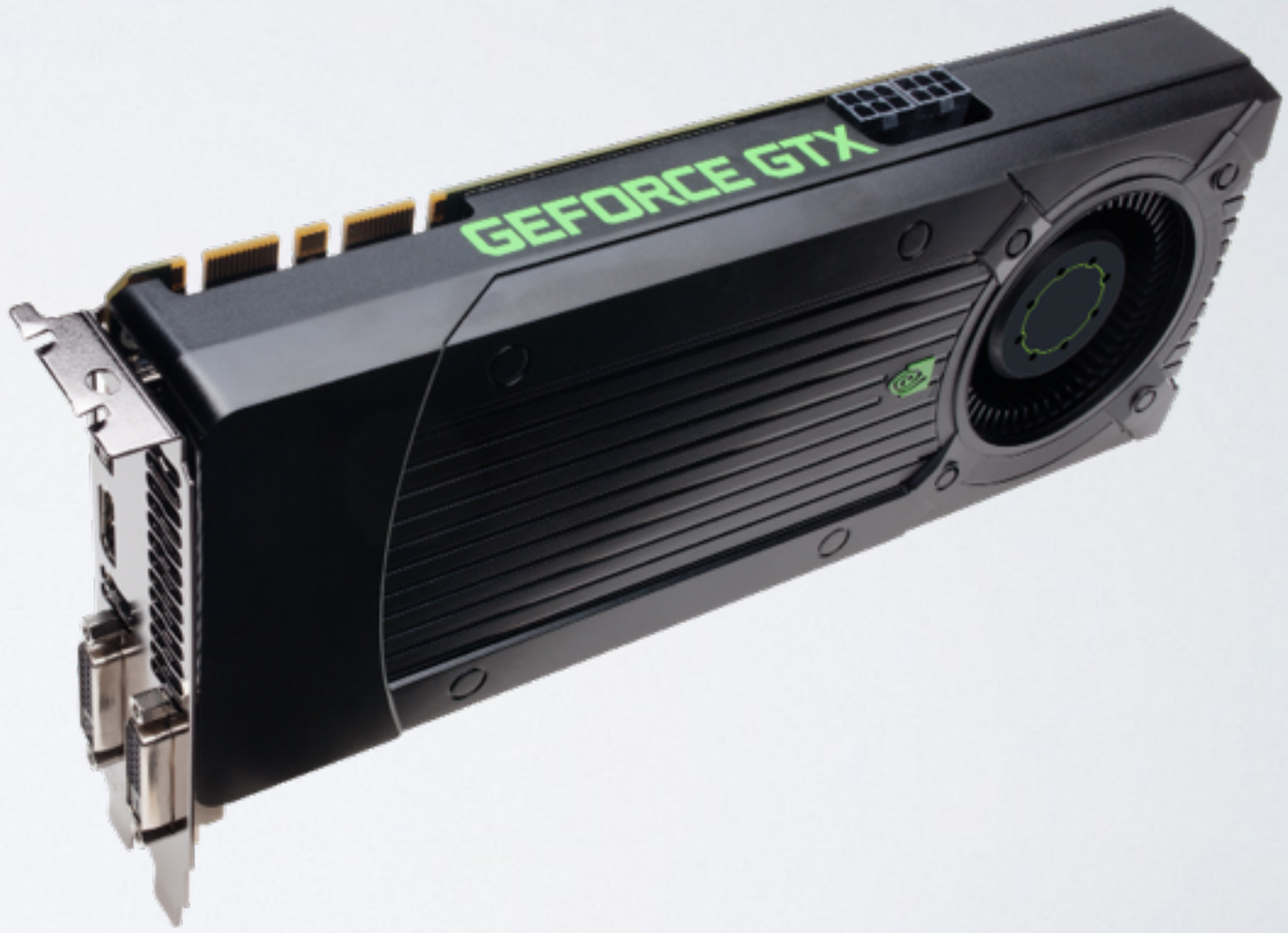
- DDR3 1600 MHz
- Fiabilidad y Alto Rendimiento

TARJETA GRÁFICA

Requisitos

- Procesamiento Paralelo CUDA

- Alto Rendimiento

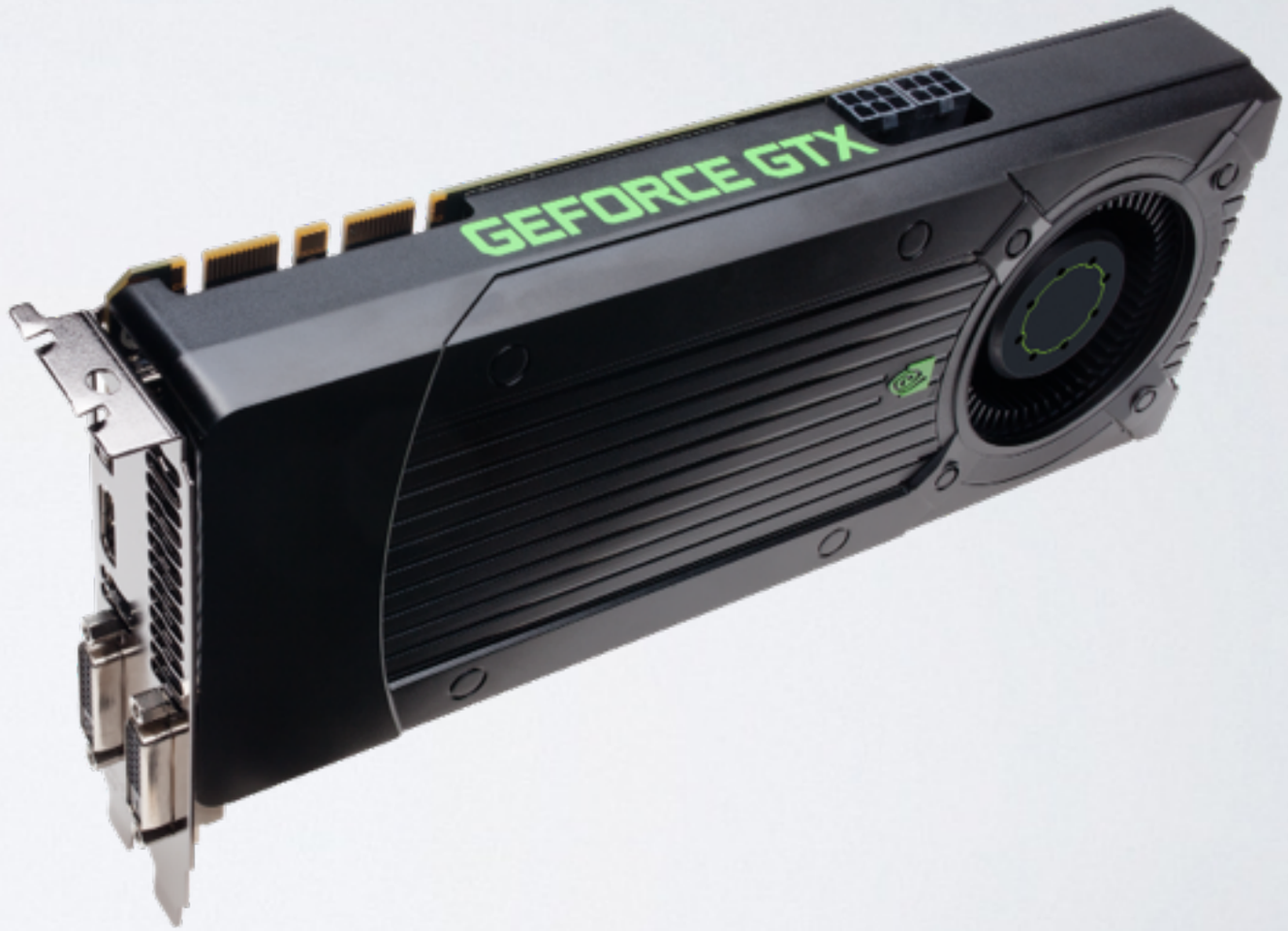


TARJETA GRÁFICA

Ofrece

- 1344 Núcleos CUDA
- 915MHz
- 2GB GDDR5
- Resolución 4K
- HDMI, DVI, Display Port
- Horas de diversión

NVIDIA GeForce GTX 660



ALMACENAMIENTO

Requisitos

- Acceso Rápido
- Bajo Consumo

ALMACENAMIENTO

Ofrece

Crucial CT240M500SSD1

- SATA3 6Gb/s
- 240 GB
- Lectura Aleatoria 72000 IOPS
  Escritura Aleatoria 60000 IOPS
- Durabilidad 5 Años 40GB/día

UNIDAD ÓPTICA

Requisitos

- Barata
- Compatibilidad

UNIDAD ÓPTICA

Ofrece

Samsung SH-224DB/BEBE

- Bajo Coste
- Velocidad Escritura Hasta x24
- Transferencia Hasta 32,4 Mb/s
- Formatos: DVD-R DC, DVD-RW, DVD+R DC, DVD+RW, DVD-ROM, DVD-Vidéo, DVD-RAM, CD-ROM, CD-R, CD-RW, CD-DA, CD+E(G), CD-MIDI, CD-TEXT…

TARJETA WIFI

Requisitos

- Barata
- Compatibilidad

TARJETA WIFI

Ofrece

- Bajo Coste
- IEEE 802.11 b/g/n
- Seguridad 64-bit WEP, 128-bit WEP, WPA2-PSK, WPA-PSK, WPS
- Hasta 300Mbps

ASUS PCE-N15

TORRE

Requisitos

- Barata
- Forma ATX
- Buena ventilación

TORRE

Ofrece

- Compatibilidad ATX
- Hasta 6 Ventiladores
- Coste moderado
- 18,5 x 43,0 x 45 cm

NOX Coolbay SX

FUENTE DE ALIMENTACIÓN

Requisitos

- Barata
- Satisfacer la Potencia del Sistema
- Conexiones suficientes
- Marca conocida

FUENTE DE ALIMENTACIÓN

Ofrece

Corsair CS Series CS650M

- 650 W de Potencia
- Media de Fallos 150000 horas
- 1xATX, 1xEPS, 4xPCI-E, 4xConector 4-Pin , 6xSATA, 1xFloppy
- 80 PLUS Gold

PRESUPUESTO

Nombre del Componente	Cantidad	Precio
Placa Base Gigabyte GA-B75-D3V http://www.pccomponentes.com/gigabyte_ga_b75_d3v.html?gclid=CJyJtcvZ5LsCFW_MtAodC38AgA	1	71 €
Procesador Intel Core i5 3340S http://www.amazon.com/INTEL-BX80637I53340S-CORE-I5-3340S-2-80GHZ/dp/B00EUXRPWY	1	151,9 €
Disipador de Calor del Procesador Enermax ETS-T40-TB http://xtremmedia.com/Disipador_Gaming_Enermax_ETS_T40_TB.html	1	30,80 €
Memoria Principal Kingston HyperX blu 2x4GB http://www.4frags.com/catalog/product_info.php?products_id=45076&name=memoria+kingston+hyperx+2x4gb++ddr3-1600+-+khx1600c9d3b1k2/8gx	1	69,8 €
Tarjeta Gráfica GeForce GTX 660 http://www.pccomponentes.com/asus_geforce_gtx_660_diretcu_ii_2gb_gddr5.html?gclid=CM7fmN_b5LsCFTHLtAodikoAzw	1	159 €
Almacenamiento Crucial CT240M500SSD1 http://www.4frags.com/catalog/product_info.php?products_id=65255&name=crucial+ssd+m500+240gb+-+ct240m500ssd1	1	134,18 €
Lector y Grabador DVD/CD Samsung SH-224DB/BEBE http://www.pccomponentes.com/samsung_sh_224db_grabadora_dvd_24x_oem_negra.html?gclid=CI_P-5zc5LsCFQPmwgodcWYAnQ	1	15,95 €
Tarjeta Ethernet Asus PCE-N15 http://www.amazon.es/ASUS-PCE-N15-Accesorio-Inalámbrico-802-11b/dp/B0053GR2YI/ref=sr_1_1?ie=UTF8&qid=1388845452&sr=8-1&keywords=Asus+PCE-N15	1	17 €
Torre NOX Coolbay SX http://www.amazon.es/NOX-NXCBAYSX-carcasa-ordenador-sintéticos/dp/B008CXLJY8/ref=sr_1_1?ie=UTF8&qid=13888455 18&sr=8-1&keywords=NOX+Coolbay+SX	1	40,59 €
Cooler Master 120x120mm 1200RPM http://www.4frags.com/catalog/product_info.php?products_id=4963&name=ventilador+cooler+master++silent+fan+120mm+-+1200rpm	3	3x3,79=11,37 €
Fuente de Alimentación Corsair CS Series C650M http://www.pccomponentes.com/corsair_cs650m_650w_80_plus_gold.html?gclid=CIP4gaCQjbwCFa-WtAodBR4AKw	1	85 €

PRECIO DEL EQUIPO

786,59€

Juan Carlos De La Torre Macías

CUESTIÓN D

Idear un programa muy sencillo y codificarlo en ARM o x86 (a elegir) para que se ejecute directamente en una máquina que tenga esa arquitectura (no …

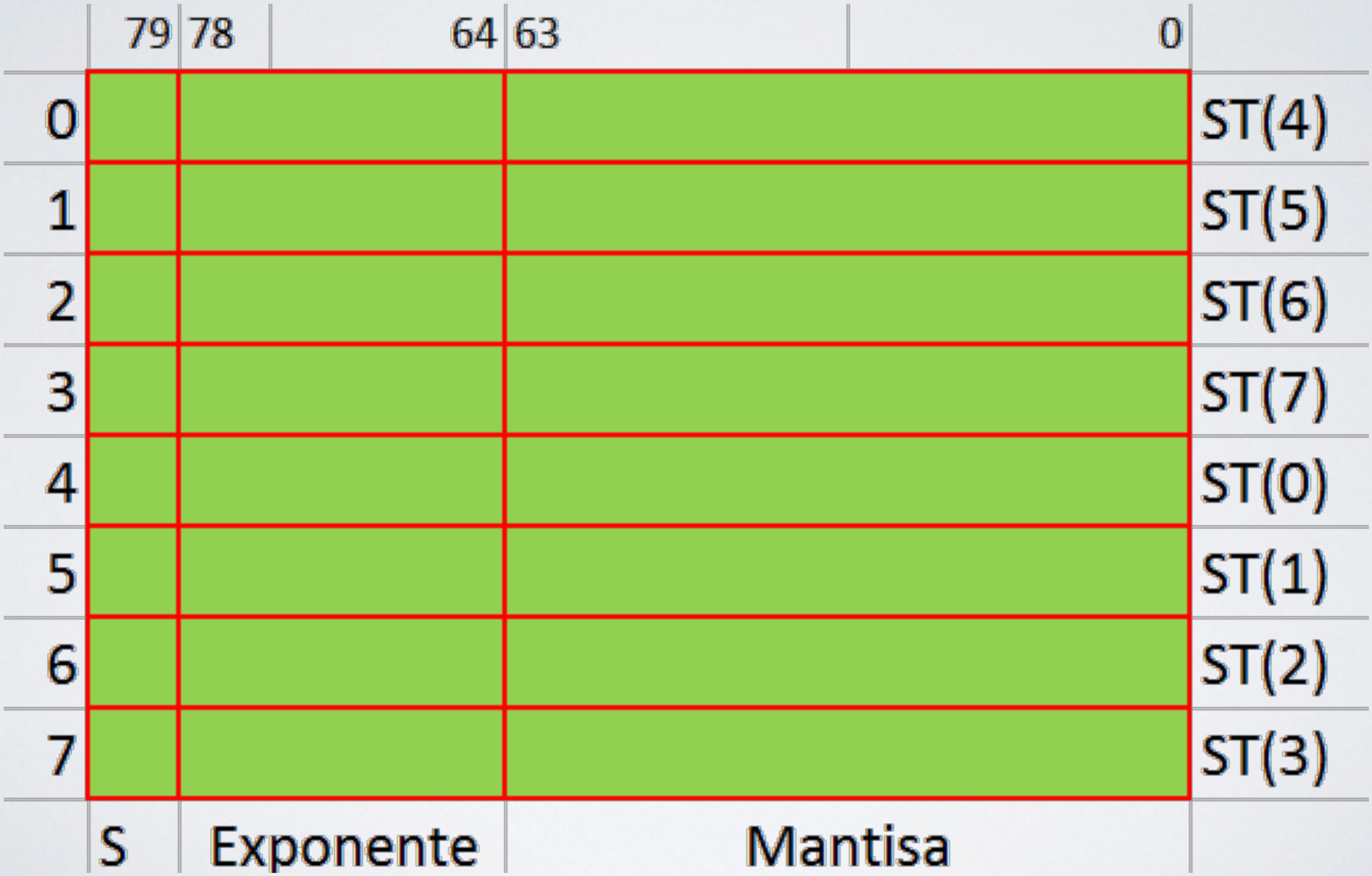
ESTRUCTURA BÁSICA

- Se trabaja con una pila circular de 8 registros para los datos.

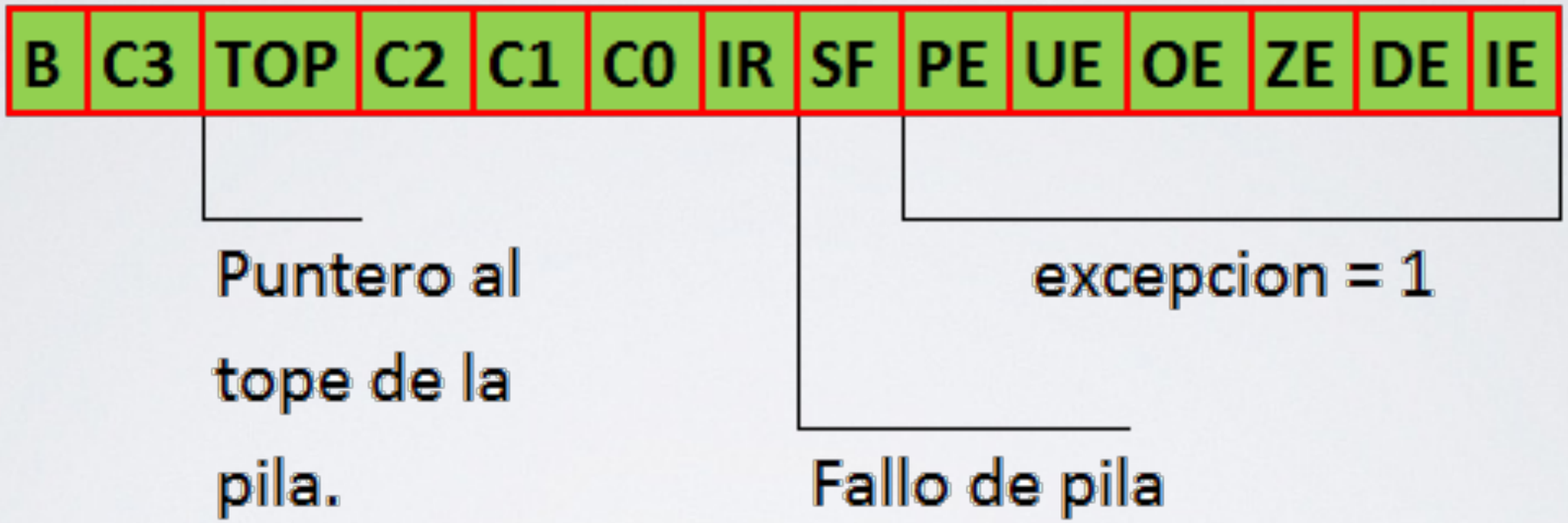
- Sobre esta pila operan las instrucciones, bien sobre un solo registro o sobre dos, depositando el dato nuevamente en la pila.

- Obviamente se puede apilar datos, operar reiteradamente sobre el tope, extraer datos, etc.

PILA DE DATOS



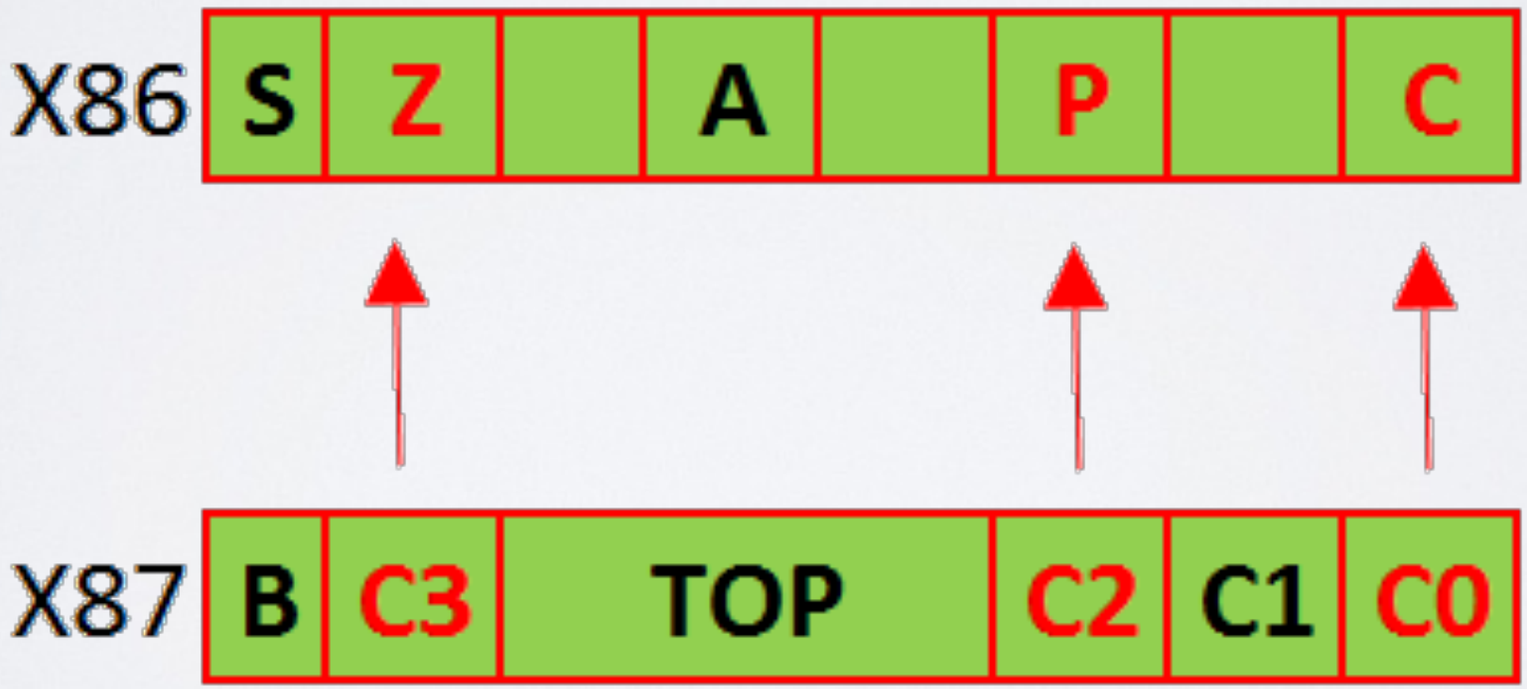
REGISTROS DE **ESTADO**, CONTROL Y MARCA



	C3	C2	C0
ST>ST1	0	0	0
ST<ST1	0	0	1
ST=ST1	1	0	0
--------	1	1	1

CORRESPONDENCIA FLAGS X86-X87

- La instrucción FSTSW AX carga en el registro AX las banderas de x87.
- La instrucción SAHF deposita en contenido de AH en las banderas de x86.



EJEMPLO DE INSTRUCCIONES...

…de CARGA

FLD op1

- Mem32
- Mem64
- Mem80
- ST(i)

…de EXTRACCIÓN

FST op1

- Mem32
- Mem64
- ST(i)

EJEMPLO DE INSTRUCCIONES...

…de CARGA

FLD op1

- Mem32
- Mem64
- Mem80
- ST(i)

…de EXTRACCIÓN

FST op1

- Mem32
- Mem64
- ST(i)

NUESTRO PROGRAMA...

...resuelve una ecuación de segundo grado.

```
start:
;Inicio del programa principal.


   Invoke puts, "--- Calculo de ecuaciones de segundo grado —"

   Invoke puts, "-------------- aX^2 + bX + c = 0 -------------"

repetir_a:

   Invoke printf, "Introduzca el valor de 'a' (a<>0): "

   Invoke scanf, "%f", Addr aa

   cmp d[aa], 0              ;comparamos el valor introducido con cero

   je > division_por_cero;Salta a la etiqueta division_por_cero si 'a'=0

   Invoke printf, "Introduzca el valor de 'b' : "

   Invoke scanf, "%f", Addr bb

   Invoke printf, "Introduzca el valor de 'c' : "

   Invoke scanf, "%f", Addr cc

   Invoke determinante       ;calculamos el determinante que nos será devuelto en la pila
                             ;(Pila:P0=determinante)

   fldz                      ;Introducimos un cero (Pila:P0=0, P1=determinante)

   fcomi st0, st1            ;comparamos el determinante con cero (Pila:P0=0,
                          ;P1=determinante)
```

```
je > una_solucion       ;Saltamos si el determinante vale cero
ja > raiz_compleja      ;Saltamos si el determinante es negativo
                        ;Seguimos si la ecuación tiene dos soluciones
fincstp                 ;desplaza una posición la pila (Pila:P0=determinante, P1=0)
Invoke puts, "La ecuación tiene dos soluciones."
mov esi, 0              ;Inicializamos ESI a cero para que nos sirva como contador

fsqrt                   ;calculamos la raiz cuadrada del determinante
                        ;(Pila:P0=raiz_determinante, P1=0)

fst q[deter]            ;almacenamos la raiz cuadrada del determinante

finit                   ;inicializamos la pila flotante (Pila vacía)

fld q[deter]            ;introducimos la raiz del determinante en la pila
                        ;(Pila:P0=raiz_determinante)
```

```
soluciones:
    fld d[bb]           ;introducimos en la pila el valor de 'b' (Pila:P0=b,
                        ;P1=raiz_determinante)
    fchs                ;cambiamos el signo a 'b' (Pila:P0=-b, P1=raiz_determinante)

    fmul                ;multiplicamos -b*determinante (Pila:P0=-b*raiz_determinante)

    fld d[aa]           ;introducimos en la pila el valor de 'a' (Pila:P0=a, P1=-
                        ;b*raiz_determinante)

    fld q[dos]          ;introducimos un 2 en la pila (Pila:P0=2, P1=a, P2=-
                        ;b*raiz_determinante)

    fmul                ;multiplicamos 2*a (Pila:P0=2*a, P1=-b*raiz_determinante)

    fdiv                ;dividimos (-b*determinante)/(2*a) (Pila:P0=(-b*raiz_determinante)/
                        ;(2*a))

    fst q[solucion] ;almacenamos la solucion en memoria

    Invoke printf, "x=  %f ", [solucion]

    inc esi             ;sumamos 1 al contador

    cmp esi, 2          ;comprobamos si el contador vale 2

    je > fin            ;salta cuando ESI valga 2

    finit               ;inicilizamos la pila flotante(Pila vacía)

    fld q[deter]        ;introducimos la raiz del determinante en la pila
                        ;(Pila:P0=raiz_determinante)

    fchs                ;cambiamos el signo al determinante (Pila:P0=-raiz_determinante)

    jmp soluciones      ;salta para calcular la segunda solución
```

```
division_por_cero:
   Invoke puts, "--- El valor de 'a' no puede ser cero ---"
   Jmp < repetir_a ;Volvemos a pedir que introduzcan el valor de 'a'
una_solucion:
   finit           ;Inicializamos la pila flotante
   fld d[bb]       ;introducimos en la pila el valor de 'b' (Pila:P0=b)
   fchs            ;cambiamos el signo a 'b' (Pila:P0=-b)
   fld d[aa]       ;introducimos en la pila el valor de 'a' (Pila:P0=a, P1=-b)
   fld q[dos]      ;introducimos un dos en la pila
   fmul            ;multiplicamos 2*a (Pila:P0=2*a, P1=-b)
   fdiv            ;dividimos -b/(2*a) (Pila:P0=-b/(2*a))
   fst q[solucion] ;almacenamos la solucion en memoria
   Invoke printf, "--- La ecuación tiene una única solución: %f", [solucion]
   jmp > fin
raiz_compleja:
   Invoke puts, "--- La ecuación no tiene solución en el conjunto de los números reales."
fin:
```

```
;Subprogramas:

determinante:
    finit               ;inicializamos la pila
    fld d[bb]           ;almacenamos en la pila el valor de b (Pila:P0=b)
    fld d[bb]           ;almacenamos otra vez el valor de b (Pila:P0=P1=b)
    fmul                ;multiplicamos b*b. (Pila:P0=b*b)
    fld q[cuatro]       ;almacenamos en la pila el 4 (Pila:P0=4, P1=b*b)

    fld d[aa]           ;almacenamos en la pila el valor de a (Pila:P0=a, P1=4, P2=b*b)
    fld d[cc]           ;almacenamos en la pila el valor de c (Pila:P0=c, P1=a, P2=4, P3=b*b)
    fmul                ;multiplicamos a*c (Pila:P0=a*c, P1=4, P2=b*b)
    fmul                ;multiplicamos (a*c)*4 (Pila:P0=4*(a*c), P1=b*b)
    fsub                ;restamos (b*b)-(4*(a*c)) (Pila:P0=(b*b)-(4*(a*c))
    fst q[deter]        ;guardamos el resultado en memoria
ret
```

GoBug CODE BREAKPOINT RE...

file action inspect settings view help

log

Running to starting address
SHIMVIEW: ShimInfo(Complete)
Starting address reached
code breakpoint reached at 4011B1h

```
40119F: PUSH 4031A3
4011A4: CALL system(msvcrt.dll)
4011A9: XOR EAX,EAX
4011AB: PUSH EAX
4011AC: CALL ExitProcess(KERNEL32.dll)
~~~determinante:~~~>
4011B1: WAIT
4011B2: FINIT
4011B4: FLD D[bb]
4011BA: FLD D[bb]
4011C0: FMULP 1,0
4011C2: FLD Q[cuatro]
4011C8: FLD D[aa]
4011CE: FLD D[cc]
4011D4: FMULP 1,0
4011D6: FMULP 1,0
4011D8: FSUB 1,0
4011DA: FST Q[deter]
4011E0: RET
          .
..... no code .....
```

C:\Users\Emilio\Dropbox\Grupo Estudio\AC\E

```
--- Calculo de ecuaciones de segundo gra

------------- aX^2 + bX + c = 0 -------

Introduzca el valor de 'a' (a<>0): 2
Introduzca el valor de 'b' : 8
Introduzca el valor de 'c' : 3
```

Floating point registers (80-...

```
Control word: 027Fh  PC=2(53-bits) RC=0(near)
       masks: P=1 U=1 O=1 Z=1 D=1 I=1
Status word: 0000h SP=0  Condition codes=0000
      flags: S=0 P=0 U=0 O=0 Z=0 D=0 I=0  =0

ST  Exp      Mantissa          Value
0  -3FFE  +0000000000000000   -- empty --
1  -3FFE  +0000000000000000   -- empty --
2  -3FFE  +0000000000000000   -- empty --
3  -3FFE  +0000000000000000   -- empty --
4  -3FFE  +0000000000000000   -- empty --
5  -3FFE  +0000000000000000   -- empty --
6  -3FFE  +0000000000000000   -- empty --
7  -3FFE  +0000000000000000   -- empty --
```

Gracias por vuestra atención…
¿Alguna pregunta?