Arquitectura CISC vs RISC (Intel x86-64)

Luis Eduardo Henao Padilla
Febrero 2020

Univalle Sede Tuluá
Ingeniería de Sistemas
Arquitectura De Computadores 2
Marco Antonio Fula

Tabla de Contenidos

Lista de figuras

Objetivos

General:

En el siguiente trabajo escrito se busca establecer una diferenciación clara y simple entre la Arquitectura CISC y la RISC, enfocándonos mayoritariamente en las ventajas y desventajas de la arquitectura CISC, específicamente la arquitectura Intel 64 utilizada en ciertos modelos de procesadores de la familia Intel.

Específico:

Poder dar ejemplos concisos sobre cómo funciona la arquitectura CISC en lenguaje Ensamblador, y a partir de esto relacionarlo de manera directa o indirecta con características, ventajas o desventajas de dicha arquitectura.

A) Introducción:
Arquitectura CISC frente a la RISC

Uno de los factores importantes a tener en cuenta antes de diseñar un microprocesadores es decidir que juego de instrucciones usará, ya que esto determinara el lenguaje que el procesador entenderá al momento de ejecutar un programa; actualmente existen diferentes filosofías de diseño para la creación de los microprocesadores, hoy hablaremos de 2 de ellas, la arquitectura CISC y RISC.

1. ¿Qué es la Arquitectura CISC?

En arquitectura de Computadores, **CISC (**Complex instruction Set Computer o en español Conjunto de Instrucciones Complejas**)** es un tipo de arquitectura de computador, que se caracteriza por tener un conjunto de instrucciones muy amplio pero bastante lentas a la hora de ejecutarse; A su vez permite operaciones complejas entre operandos que se encuentren en memoria o en los registros internos del procesador.

2. ¿Qué es la Arquitectura RISC?

En arquitectura de computadores, **RISC (**Reduced Instruction Set Computer o en español Conjunto de Instrucciones Reducidas) es un tipo de arquitectura usado generalmente en microprocesadores o en microcontroladores, una de las características más significativas de RISC es que promueve conjuntos de instrucciones pequeños y muy primitivas que pueden tomar poco tiempo en ejecutarse.

3. Características de la Arquitectura CISC:

- Instrucciones a bajo nivel muy complejas.
- Muchos Modos de direccionamiento.
- Juego de instrucciones muy extenso.
- El tamaño de las Instrucciones que opera es variable.
- En la actualidad, generalmente son micro programados.
- Muchas instrucciones pueden acceder a Memoria.

4. Ventajas de CISC frente a RISC:

- El compilador tiene que hacer muy poco trabajo para traducir un código de alto nivel a uno de bajo nivel.
- Utiliza pocos registros

-Permite implementar instrucciones de alto nivel directamente con un número pequeño de instrucciones
- Repertorio de instrucciones complejas (Mover grandes Bloques de memoria, operaciones complejas con punto flotante, raíz cuadrada)
- Gran variedad de tipos de datos y de Modos de direccionamiento.

5 .Desventajas de CISC frente a RISC:

- Es difícil obtener el paralelismo entre instrucciones
- unidad de control más compleja
- Deficiencia energética debido a que una instrucción puede llevar a cabo varios ciclos de máquina.
- Inclusión de instrucciones que rara vez se usan.
- El tamaño de instrucciones es variable.

6. CISC en la actualidad:
En la actualidad, la mayoría de los Procesadores de 64 bits que basan su diseño en la arquitectura CISC implementan un sistema que convierte las instrucciones complejas a varias instrucciones simples del tipo RISC, llamadas micro instrucciones.

Por lo que no es erróneo afirmar que en la actualidad aún persisten los procesadores CISC pero buscan optimizar y mejorar el rendimiento del procesador mediante el uso de instrucciones tipo RISC, lo que resulta en un sistema híbrido, por decirlo de alguna manera, en el que se aprovechan ambos tipos de arquitectura.

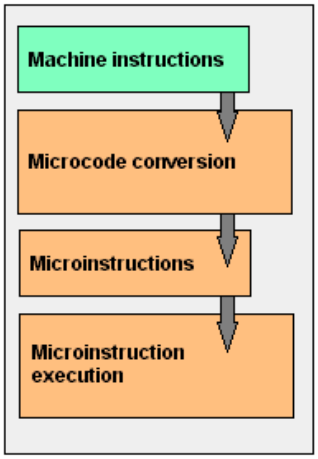

Imagen 1. Conversión de instrucción compleja a Micro Instrucciones simples.

B) Arquitectura Intel 64 (x86 – 64): CISC y sus Diferencias frente a RISC

En la siguiente parte del trabajo escrito se dará a conocer información sobre la arquitectura Intel 64 (que está basada en la arquitectura CISC) implementada en ciertos modelos de la familia Intel (Por Ej. Core 2 Duo, Intel Pentium Dual Core, Intel Atom, etcétera.).

Pero, a medida que se conozca cierta información importante o relevante de la arquitectura, se hará un paréntesis para relacionar lo escrito en la parte A) del trabajo y complementarlo, expandirlo o solo relacionarlo.

1. Definición de la Arquitectura Intel 64:

La arquitectura Intel 64 no es más que la implementación Intel de la tecnología x86 – 64 en sus procesadores. La tecnología x86 -64 es la versión de 64 bits del **conjunto de instrucciones** x86 (osea el conjunto de instrucciones de 32bits) creado o iniciado por Intel.
Cabe destacar que **x86** representa o reagrupa a los microprocesadores compatibles con el juego de instrucciones del Intel 8086 (Que es un microprocesador de 16 bits).

2. Descripción de la arquitectura:

Según la página de Intel® 64 Architecture de Intel (ver lista de referencias punto 1), la arquitectura Intel 64 proporciona soporte Para:

- Direccionamiento Virtual de hasta 64 bits
- Punteros de 64 bits
- Registros de uso general de 64 bits.
- Soporte de números enteros de 64 bits de tamaño
- Hasta un Terabyte (TB) de capacidad de direccionamiento de memoria.

3. Ensamblador x86-64:

La arquitectura x86 – 64 fue desarrollada por AMD principalmente bajo el nombre de AMD64. Pero Intel adoptó y modificó esta arquitectura y la nombró Intel 64, en este punto del trabajo estará dedicado exclusivamente en la versión x86-64 de Intel.

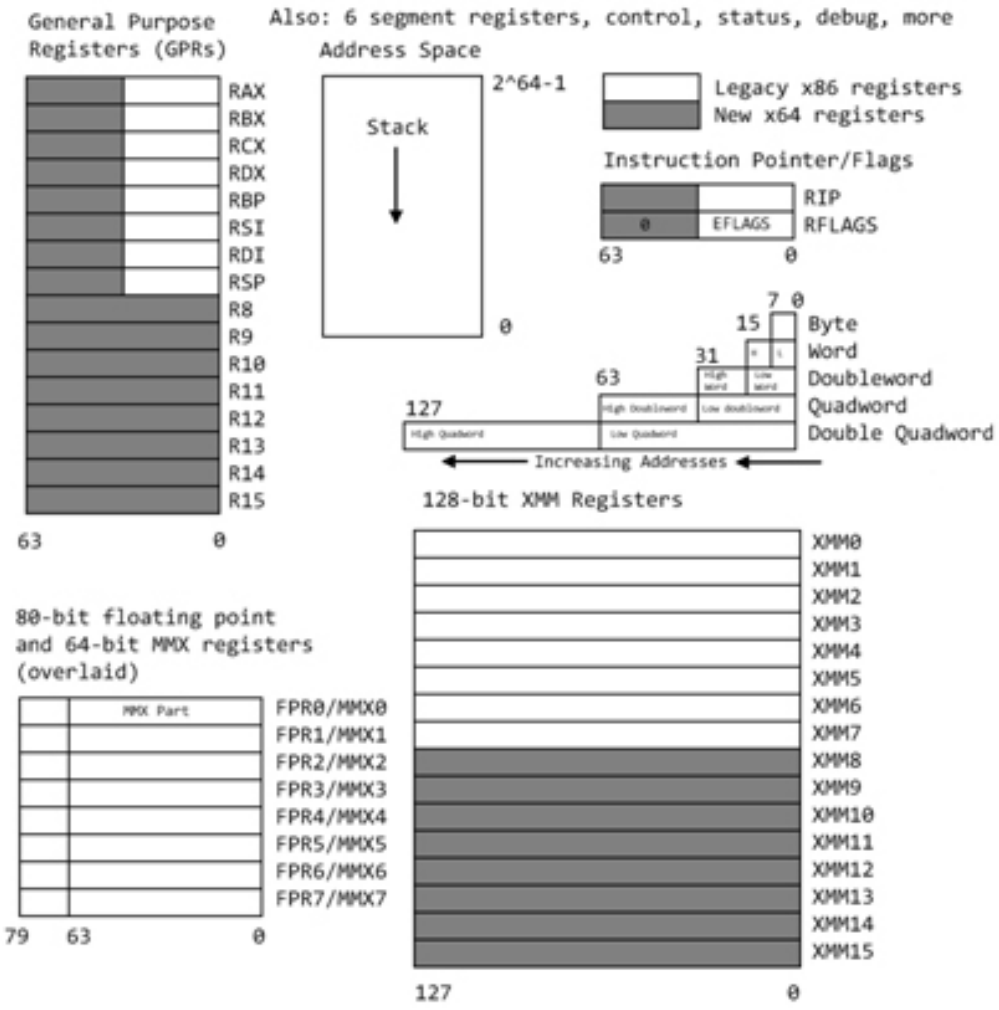

Imagen 2. Arquitectura general Intel 64

3.1. Registros:

Los procesadores de la arquitectura x86 – 64 disponen de registros de 64 bits y un espacio de direccionamiento de 64 bits como se mencionó anteriormente.

Como se puede observar en la **imagen 2** los procesadores de 64 bits cuentan con **16 registros de propósito general (**RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8, R9, R10, R11, R12, R13, R14, R15); Es importante aclarar que Intel utiliza prefijos y sufijos para referirse al tamaño de sus registros de propósito general, por lo que es importante definir:

Para los primeros 8 Registros:

Sufijo X: Registro de 16 bits. Ej: AX (2 bytes – 1 Word)
Sufijo L: Parte baja de un registro de 16 bits. Ej: AL (1 byte)
Sufijo H: Parte Alta de un registro de 16 bits. Ej: AH (1 byte)

Prefijo E: Registro de 32 bits. Ej: EAX (4 bytes – 1 DoubleWord)
Prefijo R: Registro de 64 bits. Ej: RAX (8 bytes – 1 QuadWord)

Para los 8 Registros Restantes:

Prefijo R: Registro de 64 bits. Ej: R8 (8 bytes – 1 QuadWord)
Sufijo D: Registro de 32 bits (lower Dword) Ej: R8D (4 bytes – 1 DoubleWord)
Sufijo W: Registro de 16 bits (lower Word) Ej: R8W (2 bytes – 1 Word)
Sufijo B: Registro de 8 bits (lower Byte) Ej: R8B (1 Byte)

Si seguimos analizando la **imagen 2** podremos encontrar el **Puntero de instrucción RIP** cuya función es apuntar a la siguiente instrucción a ejecutar y como se puede observar es un registro de tamaño de 64 bits.

El registro **RFLAGS** es el que almacena las banderas usadas para resultados de operaciones y para el control del procesador, La parte alta de este registro (ósea los bits de 32- 63) están reservados y actualmente sin usar, mientras que la parte baja contiene las banderas a usar, entre las más comunes tenemos: **CF, PF, AF, ZF, SF, OF, RF,** etcétera.

Los 16 registros **XMM** son de 128 bits cada uno, permiten realizar operaciones en paralelo y en si en cada registro se pueden trabajar múltiples datos en una sola instrucción (Verlo como un vector o una matriz). Los registros **MMX (**de 64 bits**)** también se pueden trabajar de la misma forma, aunque generalmente se utilizan para guardar la mantisa del numero punto flotante, estos registros Junto a 16 bits en la parte superior corresponde a un **Registro Punto Flotante** (**FPR)** respectivamente para cada registro.

Existen otros registros que son los **registros de propósito Específico** dentro de estos encontramos **CS** (Segmento de código), **DS** (Segmento de datos), **SS** (Segmento de pila), **ES, FS, GS** (Segmentos Extras); existen otros registros que son para control, memoria, debug, visualización, etcétera.

Es importante observar que se cumple una de las ventajas de la arquitectura CISC en la información anterior, y es que Intel 64 maneja una cantidad mínima de registros (de propósito general) comparados con la arquitectura RISC. Es más,

respecto a arquitecturas pasadas como la x86, estos registros (De propósito general) solo han aumentado el doble en cantidad.

3.2. Modos de direccionamiento:

Los modos de direccionamiento son las diferentes maneras de especificar un operando dentro de una instrucción en el lenguaje ensamblador, especifica la forma de calcular la dirección efectiva de un operando mediante el uso de información contenida en los registros y/o constantes.

Los más comunes dentro de esta arquitectura son:

- Direccionamiento Inmediato: el valor se almacena en la instrucción.

Ejemplo: ADD RAX, 14.
Otro ejemplo: MOV RAX, 0102030405060708h

- Direccionamiento por registro (registro a registro): Ejemplo ADD R8L, AL

- **Direccionamiento Indirecto:** Esto permite usar un desplazamientos de 8, 16 o 32 bits, para cualquier registro de propósito general e índice, y una escala de 1, 2,4 u 8 para multiplicar el índice.

Ej: MOV R8W, 1234[8*RAX+RCX], que significa: Mover la palabra (Word) en la dirección 8*RAX+RCX+1234 en el registro R8W

También es válido: MOV R8W, [8*RAX+RCX+1234]

Existe otros modos de direccionamiento como por ejemplo: **Direccionamiento a pila, Relativo, Relativo a PC, directo a memoria, Indexado**, etc. Y sin contar que el direccionamiento Indirecto y otros modos pueden encontrarse escritos de muchas formas, o por decirlo de otra manera, existe diversas formas legales de interpretar un modo de direccionamiento.

Aquí podemos observar una de las características y ventajas de la arquitectura CISC y es que estas cuentas con diversos modos y formas de acceder a un operando de una instrucción que se encuentra en memoria.

3. 3. Set de instrucciones

El juego de instrucciones de los procesadores x86-64 es muy amplio y extenso en cuanto a posibilidades para el programador, dentro de las más comunes podemos encontrar:

- **Instrucciones de transferencia de Datos:**
  - MOV destino, fuente (Mueve a destino lo que está en fuente)
  - XCHG destino, fuente (intercambia el contenido de ambos operandos)
  - PUSH fuente (Mueve el operando al tope de la pila del programa en memoria)
  - POP destino (Mueve el dato que está en el tope de la pila a destino)

- **Instrucciones aritmético – lógicas:**
  -ADD destino, fuente (suma aritmética de los dos operandos)
  -SUB destino, fuente (resta aritmética de los dos operandos)
  -DEC destino (Decrementa en una unidad el operando)
  -XOR destino, fuente (Operación or exclusiva entre los dos operandos)
  -TEST destino, fuente (Operacion And entre los dos operandos pero no guarda el resultado en ninguno de ellos)

- **Operaciones de desplazamiento:**
  SHR destino, cantidad (Desplazamiento lógico de bit a la derecha)
  ROR destino, cantidad (Rotación lógica de bit a la derecha)

- **Instrucciones de ruptura de secuencia:**
  JMP etiqueta (Salta a la posición de la etiqueta)
  JLE etiqueta (Salta si es menor o igual)
  JNC etiqueta (Salta si no hubo acarreo)
  CALL etiqueta (Llamada a subrutina)
  RET (Retorno de subrutina)
  IRET (Retorno de subrutina de servicio de interrupción RSI)

- **Instrucciones de entrada y salida:**
  IN destino, fuente (lectura de un puerto de entrada y salida)
  OUT destino, fuente (escritura a un puerto de entrada y salida)

➢ A pesar de la gran diversidad o variedad de índices o instrucciones que vimos, aún faltan una cantidad considerablemente enorme, **y aquí podemos destacar otra de las características de la arquitectura CISC** y es que en estás se manejan una considerable cantidad de Conjuntos de

instrucciones, que por motivos del objetivo de trabajo, solo se presentan las más usadas y simples.
Si el lector desea conocer que tan enorme o que tan grande es el juego de instrucciones de esta arquitectura, a continuación proveeré un enlace de un usuario de Internet llamado **Felix Cloutier** que recopiló parte de estas instrucciones para las arquitecturas **x86** y **AMD64**, basándose en la documentación de **Intel 64.**

Felix Cloutier, 2019, *x86 and amd64 instruction reference,*
Disponible en: https://www.felixcloutier.com/x86/

4. ICC (Intel C++ Compiler).

El lenguaje utilizado para programar a bajo nivel es el **ensamblador**, pero para facilitar el desarrollo de programas y ciertas operaciones de Entrada/Salida con los dispositivos utilizamos **lenguajes de alto nivel (Ej. c++)** es requerido un programa que pueda traducir el lenguaje de alto nivel a bajo nivel, es decir **un Compilador.**

4.1. Descripción:
Intel C++ Compiler es un conjunto de compiladores para los lenguajes de C y C++ desarrollados por la empresa Intel. Están disponibles para los Sistemas Operativos de Microsoft Windows, Linux y Mac Os X.
Estos compiladores pueden funcionar sobre procesadores **IA-32, Intel 64, Itanium2** y otros procesadores ajenos a la Empresa Intel, como por ejemplo **AMD**.

4.2. Ejemplos.

La idea central de este módulo es presentar una serie de ejemplos en el lenguaje c++ y su “Traducción” al lenguaje Ensamblador, y a partir de estos ejemplos poder mostrar características, Ventajas y Desventajas restantes que tiene las arquitecturas Intel 64 (basadas en el modelo CISC). Para los ejemplos se usará el **compilador x86 – 64 ICC 19.0.1.**

Ej 1. Variables de diferente tamaño Inicializadas (Ver Anexos**):**

En el siguiente ejemplo se puede observar en el código c++ que hay dos variables inicializadas, donde **x** es de tipo entero (int), z es de tipo entero corto (short) e **y** es de tipo entero largo (long). Si observamos el código en lenguaje ensamblador, podemos observar que de la linea 2 a la 4 se reserva un espacio de pila en para el programa en la memoria, a partir de las lineas 5 y 7 **podemos observar 3 aspectos de la arquitectura CISC:**

El Primer aspecto corresponde a que muchas instrucciones pueden acceder a memoria, ya sea para hacer una operación aritmética o lógicas entre algo en registro y algo en memoria, o en este caso para guardar el valor de una constante en la pila del programa que se encuentra memoria; también es importante tener en cuenta que existen diferentes instrucciones para mover datos, ya sea a registros generales o específicos, o memoria como lo son MOV, MOVS, MOVSS, MOVSW, MOVSX y un gran etcétera (lo que soporta que existan muchas instrucciones que puedan acceder a memoria).

El Segundo aspecto a resaltar de la arquitectura CISC es que se permite implementar instrucciones de alto nivel directamente con un número pequeño de instrucciones, como se puede observar en el ejemplo 1 es que las diferencias de cantidad de instrucciones entre el lenguaje a alto nivel y el de bajo nivel son muy mínimas; a medida que el programa a alto nivel vaya obteniendo cierta complejidad puede que el tamaño comience a ser más desigual, pero es importante resaltar el excelente trabajo que hace el lenguaje ensamblador a pesar de ser de bajo nivel.

El Tercer aspecto a resaltar es que a pesar de que en la línea 5, 6 y 7 se cumpla la misma instrucción de mover un número a una dirección de la pila, ambas en lenguaje maquina tendrán diferentes tamaños, debido a las diferencias de sus tipos (short, int, long), lo que hace que ocupen más o menos bytes al ser representados en lenguaje máquina.

Ej 2. Raíz cuadrada de 2 (Ver anexos **):** En el siguiente ejemplo podemos analizar que en primer lugar para el código en C++ se está implementando la librería **math.h** que se utiliza comúnmente para operaciones complejas: sen(x), cos(x), tan(x) y en nuestro caso **sqrt(2)**, a pesar de que en C++ el código es muy corto en lenguaje ensamblador este toma demasiadas líneas de código, tanto para la función f (línea 14 a la 26), como para la rutina que hace la operación (línea 1 a la 12).

Si analizamos **la linea 8** del código en ensamblador podremos darnos cuenta de algo y es que como tal, **el indice SQRTSD** (Compute Square Root of Scalar Double-Precision Floating-Point Value) es el que se encarga de realizar la raíz cuadrada de 2, **y es aquí donde podemos observar otra ventaja de las arquitecturas CISC** y es que dichas arquitecturas tienen un repertorio de instrucciones muy complejas como lo es en este caso el computo/cálculo de la raíz cuadrada de un número punto flotante de doble precisión (64 bits); otra instrucción compleja, por ejemplo es la que encontramos en la línea 7, con el índice **CVTSD2SS** que convierte un entero de palabra doble (2Word – 32 bits) en valor punto flotante de precisión doble (64 bits).

C) Conclusiones Finales

- Por más compleja o extendida que sea una función o instrucción en las arquitecturas basadas en CISC, al final el procesador tendrá que procesar lo más básico, es decir lo primitivo, puede que le tome muchos ciclos de instrucción y ejecución, pero se obtiene un resultado, a partir de lo básico, por lo que al final todo termina siendo RISC (En el sentido de Instrucciones Simples/primitivas).

- Tanto RISC Y CISC tienen sus cosas buenas y sus cosas malas, ya sea por su eficiencia, precio, instrucciones, cantidad de ciclos por operación; pero, en la actualidad esté ya es un tópico viejo, actualmente Empresas como Intel implementan tecnología RISC en sus procesadores basados en CISC para ocultar ciertas Falencias, por lo que el uso de ambas filosofías de diseño en los procesadores es el camino que se debe seguir.

D) Anexos

C++:

```
1  int f(){
2      int x=0;
3      short z=0;
4      long y=1;
5  }
```

x86 – 64 ICC 19.0.1. –> Ensamblador:

```
1  f():
2          push       rbp
3          mov        rbp, rsp
4          sub        rsp, 16
5          mov        DWORD PTR [-12+rbp], 0
6          mov        WORD PTR [-16+rbp], 0
7          mov        QWORD PTR [-8+rbp], 1
8          leave
9          ret
```

Ejemplo 1. Variables de diferente tamaño Inicializadas

C++:

```
1   #include <math.h>
2
3   int f(){
4       float a= sqrt(2);
5   }
```

x86 – 64 ICC 19.0.1. –> Ensamblador:

```
1   _ZSt4sqrtIiEN9__gnu_cxx11__enable_ifIXsr3std12__is_integerIT_EE7__valueEdE6__type
2           push    rbp                                             #476.5
3           mov     rbp, rsp                                        #476.5
4           sub     rsp, 16                                         #476.5
5           mov     DWORD PTR [-16+rbp], edi                        #476.5
6           mov     eax, DWORD PTR [-16+rbp]                        #476.29
7           cvtsi2sd xmm0, eax                                      #476.29
8           sqrtsd  xmm0, xmm0                                      #476.14
9           movsd   QWORD PTR [-8+rbp], xmm0                        #476.14
10          movsd   xmm0, QWORD PTR [-8+rbp]                        #476.14
11          leave                                                   #476.14
12          ret                                                     #476.14
13
14  f():
15          push    rbp                                             #3.8
16          mov     rbp, rsp                                        #3.8
17          sub     rsp, 16                                         #3.8
18          mov     eax, 2                                          #4.14
19          mov     edi, eax                                        #4.14
20          call    _ZSt4sqrtIiEN9__gnu_cxx11__enable_ifIXsr3std12__is_integerIT_EE
21          movsd   QWORD PTR [-8+rbp], xmm0                        #4.14
22          movsd   xmm0, QWORD PTR [-8+rbp]                        #4.14
23          cvtsd2ss xmm0, xmm0                                     #4.14
24          movss   DWORD PTR [-16+rbp], xmm0                       #4.12
25          leave                                                   #5.1
26          ret                                                     #5.1
```

Ejemplo 2. Raíz cuadrada de 2

E) Lista de referencias

1. Intel, *Intel® 64 Architecture*, recuperado de: www.intel.la/content/www/xl/es/architecture-and-technology/microarchitecture/intel-64-architecture-general.html

2. Chris Lomont, *Intel Software Developer Zone: Introduction to x64 assembly*, 19/03/2012, recuperado de: software.intel.com/en-us/articles/introduction-to-x64-assembly

3. CanadaIT, *Stack overflow,* 08/01/2020, recuperado de:stackoverflow.com/questions/44299401/difference-between-mmx-and-xmm-register

4. Miquel Albert Orenga, Gerald Enrique Manonellas, *Programación en ensamblador (x86 – 64),* recuperado de: www.exabyteinformatica.com/uoc/Informatica/Estructura_de_computadores/Estructura_de_computadores_(Modulo_6).pdf

5. Felix Cloutier, 2019, *x86 and amd64 instruction reference,* Disponible en: https://www.felixcloutier.com/x86/

6. Wikipedia, *Intel C++ Compiler,* recuperado de: es.wikipedia.org/wiki/Intel_C%2B%2B_Compiler

7. Luis Ernesto, *Tecnología CISC vs RISC,* recuperado de: https://www.monografias.com/trabajos5/teccisc/teccisc.shtml

8. Estefania Mac, *La diferencia en CISC y RISC,* recuperado de: https://techlandia.com/diferencia-cisc-risc-info_291061/