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

IIC2343 Arquitectura de Computadores

Arquitecturas de Computadores

1 Motivación

El computador básico visto hasta ahora contiene todos los elementos fundamentales de un computador. Sin embargo, existen una diversa variedad de computadores distintos que, aunque mantienen este núcleo de funcionalidades en común, presenta diferencias tanto a nivel de hardware como a nivel de software.

2 Arquitecturas de Computadores

Los distintos computadores existentes, aunque similares en los elementos fundamentales, presentan diversas diferencias en sus componentes de hardware y su definición de instrucciones. Las variaciones existentes dependerán del uso en que esté enfocado el computador particular, pudiendo incluir mayores funcionalidades o mejoras en la eficiencia. En general, la arquitectura de un computador se puede caracterizar en base a dos elementos fundamentales: la **microarquitectura** y la **arquitectura del set de instrucciones o ISA** por sus siglas en inglés.

2.1 Microarquitecturas

La microarquitectura de un computador se refiere a los distintos componentes de hardware que estarán presente en un sistema computacional. Los elementos básicos de la microarquitectura de un computador se presentan a continuación:

- **Registros:** distintos computadores tendrán distinta cantidad y tamaño de registros. En el caso del computador básico visto hasta hora se tienen dos registros de 8 bits. Aumentar la cantidad de registros de un sistema permite reducir los traspasos de datos a memoria; aumentar el tamaño de los registros (y por consiguiente de la unidad de ejecución) permite realizar operaciones con números de mayor rango y precisión. Otro elemento diferenciador en el caso de los registros es la existencia de registros de propósito especial. En el caso del computador básico, el registro B es registro de operaciones y de direccionamiento, mientras que el registro A es sólo registro de operaciones.
- **Unidades de ejecución:** la unidad básica de ejecución de todo computador es la **ALU** y como tal estará presente en toda microarquitectura. Sin embargo, es posible modificar las funcionalidades de la ALU, por ejemplo eliminando los shifters, o agregando shifters con

rotación, que en el caso de un **rotate left** vuelve a incluir el bit más significativo que fue desplazado en el bit menos significativo, y de manera inversa para un **rotate right**. También es posible agregar operaciones aritméticas más complejas, como multiplicación y división. Adicionalmente, se pueden incluir otras unidades de ejecución como una *floating point unit (FPU)* e incluso varias unidades repetidas (varias ALU) por ejemplo para permitir realizar más cálculos de manera simultánea.

- **Unidad de control:** la unidad de control también puede presentar variaciones dependiendo del computador. En particular existen dos modelos de unidades de control principales: sistemas **hard-wired** en los cuales cada instrucción tiene asociado directamente las señales de control que ejecutan una función en el computador (como en el computador básico) y sistemas **micro-programados** o de **microcode**. En estos últimos sistemas, la unidad de control es de mayor complejidad, y cada instrucción del programa se traduce en un conjunto de microinstrucciones, que ejecutan subfunciones muy específicas.

- **Condition codes:** los condition codes presentes en un computador también variarán de sistema en sistema. Sistemas minimalistas trabajarán sólo con los condition codes Z y C por ejemplo. Condition codes adicionales a los del computador básico son los siguientes:

  - **Parity (P)**: indica si el resultado de la operación es par o impar, revisando el bit menos significativo (si ese bit es 1, el resultado es impar).
  - **Auxiliary carry (AC, DC o A)** indica si hubo carry en el primer *nibble* del número, es decir en los primeros 4 bits.

- **Stack:** la presencia de un stack para subrutinas también es un elemento que variará según el sistema computacional. Adicionalmente, algunos sistemas tendrán un stack manejado de manera independiente a la memoria de datos, y por tanto con una limitación de niveles de profundidad menor que si estuviese integrado en la memoria

- **Memorias:** la cantidad de palabras de las memorias y el tamaño de cada una de estas también será un elemento que variará entre distintos computadores. En el caso del computador básico visto hasta ahora, la memoria de datos tiene dirección de 8 bits es decir $2^8 = 256$ palabras, cada una de 8 bits. La memoria de instrucciones por su parte tiene dirección de 8 bits es decir $2^8 = 256$ palabras, cada una de 15 bits.

Además de estos elementos básicos, existe una variedad mayor de elementos diferenciadores que tienen que ver con mejorar la eficiencia de la comunicación entre las partes del computador, y en el procesamiento de información. Estos elementos irán siendo incluidos más adelante al modelo de computador visto hasta ahora.

Una característica relevante del computador básico respecto a sus memorias, es la presencia de dos memorias separadas una de instrucciones y una de datos. Este paradigma arquitectónico se conoce como **arquitectura Harvard**, nombrado en honor al computador Havard Mark 1, que fue diseñado con esta arquitectura. Una alternativa a esto, es tener sólo una memoria, que tenga datos e instrucciones, lo que se conoce como **arquitectura Von Neumann**, nombrada en honor al matemático John Von Neumann quien ideo esta arquitectura. A continuación se presentan las características de ambos paradigmas, contextualizados en el computador básico y en microarquitecturas reales.

2.1.1 Arquitectura Harvard

La arquitectura Harvard se caracteriza por tener dos memorias, una de instrucciones y una de datos. Esta arquitectura presenta una serie de ventajas:

- Permite tener tamaños distintos de palabras de datos y de instrucciones.
- Permite tener tamaño de direcciones distintas para memoria de datos e instrucciones.
- Permite tener tecnologías distintas en las memorias. En el caso del computador básico, por ejemplo se ocupa una ROM para la memoria de instrucciones y una RAM para la de datos.

En general, esta arquitectura se utiliza principalmente en microcontroladores o en sistemas embebidos. A continuación se presenta un ejemplo de microarquitectura Harvard, el microcontrolador PIC16F877A:

Microcontrolador PIC16F877A

Características:

- **Registros:** el PIC16F877A tiene un solo registro de trabajo para realizar operaciones, denominado registro **W**. Adicionalmente cuenta con un registro de propósito específico para realizar direccionamiento indirecto, el registro **FSR**. Ambos registros son de 8 bits.
- **Unidades de ejecución:** el PIC16F877A tiene como única unidad de ejecución una ALU de 8 bits sin shift aritmético, pero con rotate left y right (shift lógico).
- **Unidad de control:** el mecanismo de control del PIC16F877A es hard-wired.
- **Condition codes:** los condition codes presentes en el PIC16F877A son el bit cero (Z), carry (C) y carry auxiliar (AC).
- **Stack:** el PIC16F877A tiene un stack independiente de la memoria de datos, de 8 niveles y palabras de 13 bits.
- **Memorias:** la memoria de instrucciones del PIC16F877A tiene dirección de 13 bits, es decir contiene $2^{13} = 8K$ palabras, cada una de 14 bits. La memoria de datos tiene dirección de 9 bits, es decir $2^9 = 512$ palabras de 8 bits. Parte de la memoria está reservada para funciones especiales, por lo que el tamaño efectivo es de 368 palabras. Esta memoria se caracteriza por estar subdividida en 4 regiones lógicas denominadas bancos. La ventaja de esto es que ocupando un selector de banco de 2 bits es posible direccionar la memoria con sólo 7 bits.

2.1.2 Arquitectura Von Neumann

El segundo paradigma arquitectónico predominante corresponde a la arquitectura Von Neumann. En esta arquitectura, tanto instrucciones como datos están almacenados juntos en una misma memoria. La gran ventaja de esto, es la capacidad de **poder crear programas para el computador, ocupando el propio computador**. En un computador con arquitectura Harvard los programas deben ser creados externamente y escritos en la ROM con mecanismos especiales. En un computador con arquitectura Von Neumann, en cambio, al poder trabajar los programas como

si fueran datos, permite crearlos directamente en el mismo equipo, y es por esto que esta es la arquitectura predominante en computadores de uso personal.

La arquitectura de Von Neumann tiene una implementación más simple que la arquitectura de Harvard, gracias a tener una sola vía de acceso para datos e instrucciones. Una desventaja, sin embargo, es el "'cuello de botella"' que se genera debido a no poder acceder a las instrucciones y datos en paralelo. Otra desventaja pasa por la seguridad: al tener las instrucciones y datos juntos, es teóricamente posible modificar en tiempo de ejecución un programa, pudiendo insertar código malicioso. Estás desventajas, no obstante, fueron considerada menores respecto a las ventajas de simpleza y programabilidad, por lo cual esta arquitectura es la más usada hoy en día.

La microarquitectura más importante que sigue el paradigma de Von Neumann es la de los procesadores Intel. A continuación se detallan las características del primer microporcesador Intel usado en computadores personales, el Intel 80186.

Intel 80186

Características:

- **Registros:** El Intel 80186 posee 4 registros de uso general: AX, BX, CX, DX, todos de 16 bits. Para estos registros, se permite también acceder a los sub registros *high* y low que corresponden a los 8 bits más y menos significativos respectivamente. De esta forma, el registro AX, puede ser trabajado como dos registros de 8 bits, AH y AL, lo mismo para BX (BH y BL), CX (CH y CL) y DX (DH y DL). Además cuenta con registros de uso específico también de 16 bits: SP y BP, para manejo del stack; SI y DI para direccionamiento indexado; y registros especiales para acceder a distintos segmentos de la memoria: CS, DS, ES y SS.
- **Unidades de ejecución:** El Intel 80186 cuenta con una ALU de 16 bits que tiene operaciones aritméticas y lógicas básicas, así como también multiplicación, división y shifts lógicos y aritméticos.
- **Unidad de control:** el mecanismos de control del Intel 80186 es una mezcla entre hard-wired y microcode.
- **Condition codes:** el Intel 80186 tiene los siguientes condition codes: carry (C), overflow (O), cero (Z), signo o negativo (S), carry auxiliar (A) y paridad (P).
- **Stack:** El Intel 80186 tiene stack integrado en la memoria y controlado con dos registros: el stack pointer (SP) y el base pointer (BP).
- **Memorias:** El Intel 80186 tiene un bus de direcciones de 16 bits y bus de datos de 16 bits.

2.2 Arquitectura del Set de Instrucciones (ISA)

El segundo elemento que define la arquitectura de un computador es la arquitectura del set de instrucciones o ISA. Las instrucciones de un computador definirán como se deben escribir programas, y se diferenciarán por los siguientes factores.

- **Tipos de instrucciones:** Dependiendo de la ISA, existirán distintos tipos de instrucciones disponibles. En general los tipos de instrucciones mínimos que se soportan en una ISA son:

  - Instrucciones de carga
  - Instrucciones aritméticas
  - Instrucciones lógicas y shifts
  - Instrucciones de salto
  - Instrucciones de subrutina

- **Tipos de datos:** Distintas ISA pueden definir distintos tipos de datos que son soportados.
- **Modos de direccionamiento:** Además de las instrucciones, la ISA debe definir que modos de direccionamiento son soportados por el sistema para generar la dirección efectiva (**effective address**) con la que se accederá a memoria. Algunos de los posibles modos de direccionamiento son los siguientes:

  - **Direccionamiento inmediato o literal:** la forma más simple de acceder a información en un computador es almacenar el dato en la instrucción misma como un literal que puede ser cargado en un registro o ocupando como operando, lo que se conoce como direccionamiento inmediato o literal.
  - **Direccionamiento por registros**: otra forma de acceder a la información del computador es accediendo a la información almacenada en los registros

  Aunque los modos anteriores son clasificados como modos de direccionamiento, el concepto se utiliza principalmente para referirse a como acceder a la información en la memoria de datos. Distintos computadores tendrán distintos modos de acceder a las palabras de la memoria, a continuación se presentan los modos principales usados:

  - **Direccionamiento directo:** el direccionamiento directo corresponde a indicar en la instrucción la dirección de memoria específica donde se encuentra el dato. Una instrucción que carga un valor de memoria en un registro, por ejemplo, contendrá el opcode que indica la carga, y además como parámetro, la dirección de la memoria de datos desde donde se obtendrá el valor. De esta forma, se está indicando **directamente** en la instrucción desde donde se obtendrá el valor de memoria:

Memoria de instrucciones

Dirección	Opcode	Parámetro
0x00	000100	**0x02**
0x01	...	...
0x02	...	...
0x03	...	...
0x04	...	...

Memoria de datos

Dirección	Palabra
0x00	Dato 0
0x01	Dato 1
0x02	Dato 2
0x03	Dato 3
0x04	Dato 4

– **Direccionamiento indirecto por registro:** el siguiente nivel de direccionamiento corresponde a no indicar directamente la dirección sino indicar en que registro se encuentra almacenada la dirección que se utilizará para acceder a un valor de memoria. Por ejemplo, si el registro B tiene almacenado el valor $0x03$, al ocupar direccionamiento indirecto por registro con el registro B estamos diciendo que el dato de memoria que queremos ocupar es el que está almacenado en la dirección que tiene almacenado el registro B, es decir, el dato en la dirección $0x03$:

Registros

Registro	Valor
A	...
B	**0x03**

Memoria de datos

Dirección	Palabra
0x00	Dato 0
0x01	Dato 1
0x02	Dato 2
0x03	Dato 3
0x04	Dato 4

– **Direccionamiento indirecto por registro base + offset:** el direccionamiento indirecto por registro puede ser extendido de distintas maneras. Una de estas maneras incluye que la dirección para obtener el dato se calcule ocupando el valor del registro como **base** y a esa base sumarle un **offset** o desplazamiento que puede venir incluido como parámetro en la instrucción. Por ejemplo si el registro B tiene almacenado el valor $0x01$ y el parámetro de la instrucción es $0x02$, la dirección apuntada es $0x01 + 0x02 = 0x03$:

Registros

Registro	Valor
A	...
B	**0x01**

Memoria de instrucciones

Dirección	Opcode	Parámetro
0x00	000101	**0x02**
0x01	...	...
0x02	...	...
0x03	...	...
0x04	...	...

Memoria de datos

Dirección	Palabra
0x00	Dato 0
0x01	Dato 1
0x02	Dato 2
0x03	Dato 3
0x04	Dato 4

- **Direccionamiento indirecto por registro base + registro índice:** otra manera de extender el direccionamiento indirecto por registro es ocupar dos registros para almacenar la direción: un registro base y un registro que almacena el offset, conocido como registro **índice**. Por ejemplo, si el registro B se ocupa como base y tiene el valor $0x01$ y el registro A se ocupa como índice y tiene el valor $0x03$, al dirección indicada es $0x01 + 0x03 = 0x04$:

Registros

Registro	Valor
A	**0x03**
B	**0x01**

Memoria de datos

Dirección	Palabra
0x00	Dato 0
0x01	Dato 1
0x02	Dato 2
0x03	Dato 3
0x04	Dato 4

- **Direccionamiento indirecto por registro con post incremento:** el registro indirecto por registro puede ser extendido para automáticamente incrementar el valor del registro base en 1 lo que es útil para ir recorriendo una secuencia de valores relacionados de palabras de memoria, lo que se conoce como un **arreglo** de datos:

Registros

Registro	Valor actual	Valor siguiente
A	...	...
B	**0x00**	0x01

Memoria de datos

Dirección	Palabra
0x00	Dato Arreglo 0
0x01	Dato Arreglo 1
0x02	Dato Arreglo 2
0x03	Dato Arreglo 3
0x04	Dato Arreglo 4

– **Direccionamiento indirecto por registro con post decremento**: Similar al anterior, pero el valor del registro de dirección se decrementa luego de ser usado:

Registros

Registro	Valor actual	Valor siguiente
A	...	...
B	**0x04**	0x03

Memoria de datos

Dirección	Palabra
0x00	Dato Arreglo 0
0x01	Dato Arreglo 1
0x02	Dato Arreglo 2
0x03	Dato Arreglo 3
0x04	Dato Arreglo 4

– **Direccionamiento indirecto:** El direccionamiento indirecto extiende la idea del indirecto por registro, pero ocupando una palabra de memoria para almacenar la dirección. Este modo es similar al modo directo, en que se utiliza el parámetro de la instrucción para direccionar, pero la diferencia es que el parámetro indica la palabra en memoria que tiene la dirección del dato, es decir indica **indirectamente** como acceder al dato. Por ejemplo, si la instrucción tiene el parámetro $0x02$, ocupando direccionamiento indirecto indica que la palabra almacena en la dirección $0x02$ (por ejemplo $0x04$) corresponde a al dirección del dato requerido en la memoria:

Memoria de instrucciones

Dirección	Opcode	Parámetro
0x00	000101	**0x02**
0x01	...	...
0x02	...	...
0x03	...	...
0x04	...	...

Memoria de datos

Dirección	Palabra
0x00	Dato 0
0x01	Dato 1
0x02	0x04
0x03	Dato 3
0x04	Dato 4

- **Manejo del stack:** El manejo del stack variará según cada computador, lo que se ve reflejado en las instrucciones disponibles en la ISA. En general, el mínimo manejo de stack corresponde a usarlo para almacenar el valor del PC en los llamados a subrutinas. Adicionalmente, el stack puede ser usado como almacenamiento general para los registros y también como mecanismo de paso de parámetros y retorno de subrutinas.

- **Formato de la instrucción:** El formato de la instrucción específica la traducción en lenguaje de máquina de una instrucción particular del set. Dependiendo de la ISA, este formato será general para todas las instrucciones, o existirán distintos formatos específicos.
- **Palabras por instrucción:** La cantidad de palabras que se requieran para almacenar una instrucción variará en distintas ISA, y también en una misma ISA, distintas instrucciones puede tener distinto largo.
- **Ciclos por instrucción:** La cantidad de ciclos que se demora en ejecutar cada instrucción también variará en distintas ISA, y también en una misma ISA, distintas instrucciones puede tener distinto largo.

La ISA está estrechamente vinculado con el **lenguaje assembly**, pero no son equivalentes. Es posible definir varios assembly para una misma ISA, ya que elementos como las secciones de datos, uso de labels y sintáxis numérica pueden ser definidos de manera distinta en distintos assembly.

Existen dos paradigmas principales para implementar ISA: Reduce Instruccion Set Computer (RISC) y Complex Instruction Set Computer (CISC), los cuales se presentan a continuación.

2.2.1 Reduce Instruction Set Computer (RISC)

Los sets de instrucciones RISC fueron desarrollados con el objetivo de minimizar la complejidad del hardware del computador, simplificando la arquitectura y diseño de este. En general un computador con ISA RISC se califica como que tiene "énfasis en el software" ya que el hardware del computador sólo provee funcionalidades básicas, y las funcionalidades avanzadas son implementadas por software. Además de esto, una ISA del tipo RISC tiene y privilegia instrucciones *register-to-register*, con excepción de instrucciones especiales de *load* y *store* que se encargan de los accesos a memoria.

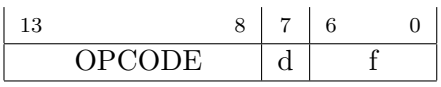
Sus características principales son las siguientes:

- Instrucciones de un sólo ciclo de clock.
- Unidad de control hard-wired
- Formato de instrucción uniforme e idealmente almacenado en sólo una palabra.
- Registros de propósito general idénticos, que permiten todos realizar las mismas funciones.
- Modos de direccionamiento simples.
- Códigos en assembly largos.
- Pocos tipos de datos soportados directamente en hardware.

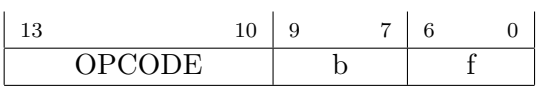
El computador básico tiene una ISA se puede clasificar como RISC, cumpliendo con todos los requisitos anteriores, salvo el hecho de tener registros de propósito general con las mismas funciones. El microcontrolador PIC16F877A también tiene una ISA que se puede clasificar como RISC. Sus características se describen a continuación.

ISA PIC16F877A

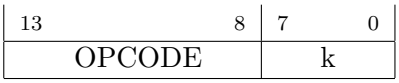
- **Tipos de instrucciones:**
  - Instrucciones de carga: Instrucciones especiales para cargar entre el registro W y memoria y para cargar un literal al registro W (ver modos de direccionamiento).
  - Instrucciones aritméticas: Instrucciones para sumar y restar.
  - Instrucciones lógicas y shifts: Instrucciones para implementar las operaciones lógicas AND, OR y XOR. Además cuaenta con instrucciones para realizar shift rotates.
  - Instrucciones de salto: El PIC tiene dos tipos de instrucciones de salto: salto con comparación en un ciclo, implementado en las instrucciones `DECFSZ f,d` y `INCFSZ f,d` que decrementan o incrementan el valor de memoria guardado en f y se saltan la siguiente instrucción si $Z = 0$; salto con bit test, implementado en las instrucciones `BTFSC f,b` y `BTFSS f,b` que se saltan la siguiente instrucción si el bit b del valor de memoria guardado en f está en 0 ("'Clear"') o están en 1 ("'Set"') respectivamente.
  - Instrucciones de subrutina:
- **Tipos de datos:** Las variable sólo pueden ser de 8 bits, es decir de 1 byte, que es el tamaño de palabras de la memoria de datos.
- **Modos de direccionamiento:** La ISA implementa los siguientes modos de direccionamiento:
  - Direccionamiento inmediato: realizado mediante instrucciones especiales de la forma "'inst"'`LW`, como por ejemplo `MOVLW k`, que copia en el registro W el valor de k. Soporta literales de 8 bits.
  - Direccionamiento directo: realizado mediante instrucciones especiales de la forma "'inst"'`WF` para mover de W a f, como `MOVWF f`, e instrucciones de la forma "'inst"'`WF` para mover de f a W, como `MOVF f,d`. Debido a que se utiliza una instrucción especial, distinta a la del literal, no se ocupan elementos de sintáxis especial (como los paréntesis) para diferenciar la dirección de memoria del valor asociado a esta.
  - Direccionamiento indirecto por registro: el direccionamiento indirecto se implementa ocupando el registro especial FSR.
- **Manejo del stack:** Sólo para almacenar PC al hacer llamados a subrutinas con `CALL`, no se puede ocupar para almacenamiento general.
- **Formato de la instrucción:** Distintos tipos de instrucción tienen distintos formatos:
  - Byte-oriented:



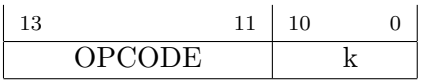
  - Bit-oriented:



– Literal:



– Saltos:



Donde:

* d representa el destino: $d = 0$ representa el registro W; $d = 1$ representa la dirección de memoria f
* f representa la dirección de memoria de datos.
* b representa un bit, comenzando en 0 desde el menos significativo.
* k representa un literal.

- **Palabras por instrucción:** Una palabra de 14 bits por instrucción.
- **Ciclos por instrucción:** Salvo algunas excepciones, las instrucciones son de 1 ciclo.

A modo de ejemplo, el siguiente código de multiplicación en lenguaje de alto nivel:

```
byte var1 = 2;
byte var2 = 3;
byte res = 0;
byte i = var1;
do
{
  res += var2;
  i--;
}
while(i != 0);
```

que se escribe así en el assembly del computador básico:

```
DATA:
    var1    2
    var2    3
    res     0
    i       0
CODE:
            MOV A, (var1)
            MOV (i), A
    start:  MOV  A,(res)
            ADD  A,(var2)
            MOV  (res),A
            MOV  A,(i)
            SUB  A,1
            MOV  (i),A
            CMP  A,0
            JNE  start
```

se escribe de la siguiente forma en el assembly de PIC16F877A:

```
var1    EQU  H'20'
var2    EQU H'21'
res      EQU  H'22'
i        EQU  H'23'

        MOVLW 2
        MOVWF var1
        MOVLW 3
        MOVWF var2
        MOVF var1,0
        MOVWF i
start:  MOVF  res,0
        ADDWF var2,0
        MOVWF res
        DECFSZ i,1
        GOTO start
```

2.2.2 Complex Instruction Set Computer (CISC)

Los sets de instrucciones CISC se caracterizan por tener muchas instrucciones y de alta complejidad, enfocándose en tener componentes de hardware específicos para implementar distintas funcionalidades. Los computadores con ISA CISC se califican con "énfasis en el hardware" ya que poseen diversos componentes de hardware para soportar instrucciones avanzadas, y habitualmente incluyen unidad de control con microcode. Además, una ISA CISC tiene y privilegia instrucciones del tipo *memory-to-memory*

Características

- Instrucciones de múltiples clock.
- Unidad de control con algún grado de microcode.
- Códigos cortos.
- Tipos de datos complejos implementados en hardware.

El ejemplo clásico de ISA es el set de instrucciones de las arquitecturas de Intel, denominado **x86**. A continuación se presentan sus características principales.

ISA Intel x86

- **Tipos de instrucciones:**
  - Instrucciones de carga: Instrucción general `MOV` para realizar transferencias ocupando distintos tipos de direccionamiento.
  - Instrucciones aritméticas: Instrucciones de suma y resta, además de instrucciones de multiplicación con y sin signo (`IMUL` y `MUL`) y división con y sin signo (`IDIV` y `DIV`).
  - Instrucciones lógicas: Instrucciones para implementar las operaciones lógicas AND, OR, XOR NOT. Además cuaenta con instrucciones para realizar shift rotates y shift normales.

  - Instrucciones de salto y subrutina: saltos condicionales, sin signo y con signo, además de saltos por excepciones (overlfow y carry). .

- **Tipos de datos:** Soporta variables del tamaño de la palabra de memoria (16 bits), lo que se conoce como tipo **word** y también variable de 1 byte (8 bits). Esto se puede debido a que los registros generales se pueden ocupar completos (e.g AX) o como dos registros de 8 bits (e.g AH y AL). Un elemento importante a considerar debido a lo anterior, es que al definir una variable de un cierto tipo (byte o word), esa variable solo se podrá almacenar en un registro de tamaño equivalente.

- **Modos de direccionamiento:** La ISA x86 soporta una gran variedad de modos de direccionamiento, los cuales no requieren instrucciones especiales.

  - Direccionamiento inmediato: se implementa indicando directamente el literal como segundo parámetro: `MOV AX, 10`
  - Direccionamiento por registros: se implementa indicando los registros fuente como segundo parámetro y destino como primer parámetro: `MOV AX, BX`
  - Direccionamiento directo: se implementa usando la sintáxis especial de paréntesis cuadrados: `MOV AX, [var1]`. En este caso la variable `var1` tiene que haber sido definida del tipo word para que la instrucción sea válida.
  - Direccionamiento indirecto por registros: se implementa usando la sintáxis especial de paréntesis cuadrados: `MOV AX, [BX]`
  - Direccionamiento indexado con registro base y offset: se implementa usando la sintáxis especial de paréntesis cuadrados e indicando con una suma los registros a usar: `MOV AX, [BX + SI]`
  - Direccionamiento indexado con registro base, registro índice y offset: se implementa usando la sintáxis especial de paréntesis cuadrados e indicando con una suma los registros a usar y el offset: `MOV AX, [BX + SI + 2]`

- **Manejo del stack y sburutinas:** Stack usado para almacenar *PC* luego de llamado a subrutinas, para carga de registros generales individuales, carga de todos los registros de propósito general (instrucciones `PUSHA` y `POPA`) del registro de status (instrucciones `PUSHF` y `POPF`) y usado para paso de parámetro.

- **Formato de la instrucción:** El formato depende del tipo de instrucción. Los formatos más relevantes son:

  - Operaciones entre dos registros o registro y memoria con direccionamiento indirecto:

Opcode	Size	Opmode	Reg Dest/Address mode	Reg Src/Address mode

  - Operaciones entre registro y literal:

Opcode	Size	Opmode	Reg Dest	Literal

– Operaciones entre registro y memoria con direccionamiento directo:

Opcode	Size	Address

– Saltos:

Opcode	Address

Donde:

* *Size* representa el tamaño del registro: $Size = 0$ representa 8 bits; $Size = 1$ representa 16 bits.
* *Opmode* representa el modo de la operación, indicando si los operandos son dos registros o un registro y un valor de memoria direccionado indirectamente.
* *Addressmode* indica que modo de direccionamiento indirecto se utilizó y con que registros.

- **Palabras por instrucción:** Variable, desde 1 hasta 2 palabras de 16 bits.
- **Ciclos por instrucción:** Variable, pero en general la mayoría de las instrucciones requiere múltiples ciclos del clock.

Como ejemplo, a continuación se escribe el mismo código de multiplicación antes descrito, ocupando la ISA x86.

```
    JMP     start

    var1    db 3
    var2    db 2
    res     db 0

    start:
            MOV CL, [var1]
    while:  MOV  AL,[res]
            ADD  AL,[var2]
            MOV  [res],AL
            SUB  CL,1
            CMP  CL,0
            JNE  while
```

A continuación se muestra un segundo ejemplo, que calcula el promedio de un arreglo de datos, basado en el siguiente código de alto nivel:

```
byte arreglo = new byte[]{6,7,3,4,5};
byte n = 5;
byte prom = 0;
byte i =0;
while(i<n)
{
  prom += arreglo[i];
  i++;
}
prom /= n;
```

```
    jmp start

    arreglo:db 6
            db 7
            db 4
            db 5
            db 3
    n       db 5
    prom    db 0

    start:
            MOV SI, 0
            MOV AX, 0
            MOV BX, arreglo
            MOV CL, [n]
    while:
            CMP SI, CX
            JGE end
            MOV DX, [BX + SI]
            ADD AL, DL
            INC SI
            JMP while
    end:
            DIV CL
            MOV [prom], AL
```

2.2.3 One Instruction Set Computer (OISC)

El concepto de RISC, de buscar una ISA minimalista para simplificar el diseño de la microarquitectura de un computador, se puede llevar a un extremo, en lo que se denomina ultra-RISC donde se busca encontrar una ISA que tenga el menor número de instrucciones, pero que le permitan seguir ejecutando todas las funcionalidades de un computador.

El mínimo de instrucciones necesarias para definir un computador completo es una, y por tanto el modelo de computador con una ISA de una instrucción se conoce como One Instruction Set Computer (OISC). Existen distintas instrucciones que se pueden ocupar para lograr un computador OISC, en particular una de ellas es: `SUBLEQ a,b,c` , la cual se interpreta como:

- Mem[a] = Mem[a] - Mem[b]
- if(Mem[a] $\leq$ 0) goto c

Con esta instrucción se pueden emular muchas de las instrucciones típicas de una ISA:
`JMP c`

- `SUBLEQ Z,Z, c` //Z es una variable que tiene almacenado un 0

`ADD a, b`

- dir0: `SUBLEQ Z,b, dir1` //Z = 0 - b = -b

- dir1: `SUBLEQ a,Z, dir2` //a = a - Z = a - -b = a + b
- dir2: `SUBLEQ Z,Z, dir3` //Z = Z - Z = 0

`MOV a, b`

- dir0: `SUBLEQ a,a, dir1` //a = a - a = 0
- dir1: `SUBLEQ Z,b, dir2` //Z = 0 - b = -b
- dir2: `SUBLEQ a,Z, dir3` //a = 0 - -b = b
- dir3: `SUBLEQ Z,Z, dir4` //Z = Z - Z = 0