# Instituto Tecnológico de Costa Rica

# Ingeniería en Computación

# Organización de Computadores y Lenguaje Ensamblador

# **Reporte de Proyecto**

# **Unidad Aritmético-Lógica (ALU)**

Luis Alonso Vega Brenes

201042592

Yáser Augusto Castillo Bravo

201042590

Santa Clara, 13 de Octubre de 2010

Objetivo:

Aprender a diseñar una unidad aritmético-lógica, capaz de realizar las operaciones básicas de suma, resta, multiplicación, comparación de números, corrimientos y otras operaciones lógicas y misceláneas que a ésta le corresponden, con datos de seis bits, utilizando los métodos de simplificación vistos en clase y analizando los resultados presentados como circuitos combinatorios. Una vez diseñados, debemos implementarlos con algún sistema digital para el manejo de diseño de circuitos.

Descripción del Proyecto

El Proyecto consiste en diseñar una estructura llamada ALU (unidad aritmética lógica) por sus siglas en inglés, para ser utilizada más adelante en el diseño de la arquitectura de un CPU (unidad central de procesamiento) por sus siglas en inglés. Esta unidad aritmética es un componente fundamental en cualquier procesador debido a que sin ella sería imposible realizar las operaciones básicas que constituyen la base de otras más complejas que a su vez funcionan como los bloques de construcción de los algoritmos que utiliza el software del equipo como recurso fundamental. La particularidad de esta ALU que debemos diseñar, es que debe implementarse con operaciones utilizando palabras o entradas de 6 bits. También debe ser capaz de realizar operaciones con signo en el caso de que se trabajen números, tomando la forma de representación conocida como complemento a dos.

Las funciones que posee el diseño de la unidad aritmética y lógica son las siguientes:

* **Sumador-Restador**: Debe tener una línea de control la cual permita seleccionar la operación deseada, en este caso la línea de control debe ser de un solo bit, ya que sólo existen dos operaciones para este circuito. También posee una línea habilitadora, la cual es la que permite realizar cualquier operación seleccionada en la línea de control, así como también las banderas de salida SF que determina el tipo de signo, ya sea positivo o negativo; ZF que retorna 1 si el valor resultado es 0, y CF, la cual indica si existe el acarreo de un bit al final de la operación.
* **Multiplicador:** Debe ser capaz de realizar multiplicaciones en complemento a dos. Posee, además de una línea habilitadora, dos entradas de 6 bits cada una, una salida de 12 bits (dada, de igual forma, en complemento a dos) y las banderas de salida SF, y ZF, definidas anteriormente en este documento.
* **Lógico:** Debe realizar las operaciones OR, AND, XOR, NOT, SHL (corrimiento lógico a la izquierda), SHRL (corrimiento lógico a la derecha), SHRA (corrimiento aritmético a la derecha). Posee la línea habilitadora, y la línea de control debe ser manejada con una entrada de 3 bits ya que son 7 operaciones diferentes las que se van a realizar. Dependiendo del tipo de operación que se realiza, se pueden considerar ambas entradas (A y B) o sólo una.
* **Comparador:** Recibe dos parámetros, los compara, y genera una salida de acuerdo con la combinación utilizada por la familia de procesadores x86\* (utilizando las banderas ZF y SF), también debe incluir una línea habilitadora para activar el circuito.
* **BitScanForward (BSF):** Este circuito revisa los bits de una entrada, iniciando en el bit 0, si todos los bits son igual a 0, la bandera ZF se hace 0, de lo contrario se hace 1, y se indica el índice del primer bit que se encontró en 1 la salida BI (Bit Index). Debe contar con la línea habilitadora, y una salida de 3 bits, BI0, BI1, BI2; las cuales indicarán el índice codificado, es decir, dónde se encontró un bit en 1, si existe; de otra forma la salida BI0 deberá ser 0. Al igual que los circuitos anteriores, se debe agregar una línea habilitadora.

Solución del problema

## Análisis de resultados

**Sumador Restador**

Entradas: Dos palabras de 6 bits, un bit de control (suma o resta), un bit de acarreo y una señal habilitadora.

Salidas: Una palabra de 6 bits con el resultado de la suma o resta, según corresponda; una señal de acarreo, una bandera de signo y otra de “cero”.

El circuito se completó con éxito.

**Multiplicador**

Entradas: Dos palabras de 6 bits y una señal habilitadora.

Salidas: Una palabra de 12 bits con el producto de la operación, una señal de signo y una bandera de “cero”.

El circuito se completó con éxito.

**Lógico**

Entradas: Dos palabras de 6 bits, tres bits de índice y una señal habilitadora.

Salidas: Una palabra de 6 bits con el resultado de la operación selecciona según el índice de entrada.

El circuito se completó con éxito.

**Comparador**

Entradas: Dos palabras de 6 bits y una señal habilitadora.

Salidas: Dos banderas (ZF y SF) que indican el resultado de la comparación numérica de las palabras de entrada según los procesadores x86.

El circuito se completó con éxito.

**BSF**

Entradas: Una palabra de 6 bits y una señal habilitadora.

Salidas: Un índice de 3 bits que indica la posición del primer bit que contiene un 1, y una bandera de “cero”.

El circuito se completó con éxito.

## Análisis de la solución

**Sumador-Restador**

El primer circuito del diseño, es el sumador-restador, el cual posee la funcionalidad de sumar. Para esto se construye un sumador de 3 bits (contando el acarreo), y a partir de ese sumador, formamos uno mayor (de 6 bits, en este caso). Además, se necesita realizar restas también, para lo cual aprovechamos las facilidades de trabajar en complemento a dos. Simplemente obtenemos el complemento a dos del número ingresado en la segunda entrada y luego seleccionamos si necesitamos la salida de ese complemento o el valor original.

Al final, buscamos cómo mostrar las banderas. Tomamos en cuenta la salida y comprobamos si es cero. Analizamos el signo de la salida (el MSB). Además, revisamos la señal de acarreo del último sumador y la mostramos como la bandera CF.

**Multiplicador**

Se llevan a cabo varios pasos principales para obtener el producto de las dos entradas numéricas representadas en complemento a dos:

* En paralelo al procedimiento descrito en los siguientes pasos, se obtiene el signo del resultado, para el cual se toman en cuenta los signos de ambas entradas.
* Se obtiene el valor absoluto de cada término ingresado, es decir, se calcula el inverso aditivo en el caso de que el número sea negativo.
* Ahora con ambos números positivos, se multiplica una de las entradas (A) con cada dígito (bit) en la otra entrada (B). Se selecciona A si el bit en B = 1, y 0 si el bit en B = 0. En otras palabras, se realiza una multiplicación bit por bit, aprovechando la facilidad que permite la base binaria.
* Se realizan los corrimientos hacia la izquierda necesarios para acomodar los resultados del paso anterior en el orden correspondiente para poder realizar la suma y obtener el resultado de la multiplicación.
* Una vez hecha la suma, se analiza el signo del resultado y se convierte a signo negativo si es necesario.

Además del producto obtenido, se comprueban las banderas necesarias, en este caso, la comprobación de cero en el resultado, revisando cada bit, y la del signo, que se encuentra en el MSB del resultado.

**Circuito Lógico**

El circuito lógico debe ser capaz de realizar distintos tipos de operaciones (And, Or, Xor, Not, Corrimiento lógico a la derecha, Corrimiento lógico a la izquierda, y Corrimiento Aritmético), con palabras dadas de 6 bits. A continuación se detalla el proceso realizado para el diseño de los circuitos mencionados.

* **AND, OR, XOR:** Para cada uno de los circuitos, es necesario evaluar los valores de 6 bits, de manera que el MSB (bit más significativo) de la primera entrada se compare con el LSB (bit menos significativo), es decir una comparación bit por bit para obtener el resultado. Esto se realiza con una caja o circuito respectivo, es decir en el OR, se realizan 6 OR bit por bit para de ahí obtener el resultado, que se obtiene uniendo las 6 salidas a un pin de salida de 6 bits que tendrá la respuesta del circuito deseado.
* **NOT:** En este circuito se recibe una sola entrada de 6 bits y se invierte cada bit para obtener el resultado.
* **SHL o Corrimiento Lógico a la Izquierda:** Siguiendo el concepto de SHL el cual menciona que en un SHL al LSB (bit menos significativo) se le asigna un 0, y el MSB es descartado, procedemos a diseñar un circuito que permita realizar esta operación, de manera que el LSB siempre es 0, y los demás bits se corren una posición adelante excepto el último bit que se pierde.
* **SHRL o Corrimiento Lógico a la Derecha:** En un corrimiento lógico a la derecha, el LSB es descartado y al MSB se asigna un 0, es decir el proceso inverso del SHL. En el cual el MSB siempre será 0, los demás bits se corren una posición hacia atrás, y el LSB se pierde o descarta.
* **SHRA o Corrimiento Aritmético a la derecha:** En el corrimiento aritmético a la derecha se replica el bit de signo, es decir el MSB, en lugar de rellenarlo con 0´s, como se hace en el proceso de SHRL, es decir el MSB se repite independientemente del valor que éste contenga, y los demás se corren una posición hacia atrás descartando el LSB.

Luego de tener estas funciones, se agregan a un único circuito que permite ingresar el índice de la función que se desea aplicar (dada en 3 bits) y las entradas (palabras de 6 bits). Una serie de circuitos selectores deciden cuál de las funciones debe activarse con cada posible entrada del índice. Finalmente se devuelve el resultado obtenido por la función correspondiente.

**Circuito Comparador**

Para resolver este problema fue necesario diseñar una tabla de verdad para el algoritmo de comparación de bits con dos entradas, esto con el fin de poder determinar un circuito base, para luego implementarlo en cascada y obtener el circuito comparador extendido y capaz de realizar operaciones con palabras de 6 bits. A continuación se detalla el proceso realizado:

**Tabla de Verdad:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A | B | X (A < B) | Y (A = B) | Z (A > B) |
| 0 | **0** | **0** | **1** | **0** |
| 0 | **1** | **1** | **0** | **1** |
| 1 | **0** | **0** | **0** | **0** |
| 1  Tabla1. Tabla de verdad de las funciones de un comparador de 2 bits | **1** | **0** | **1** | **0** |

**A y B son las entradas, X, Y, Z son las salidas para cada comparación.**

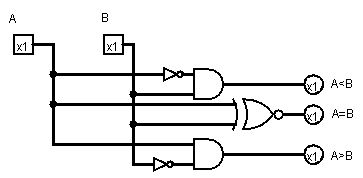
Las funciones entonces quedarían de la siguiente manera:

X (A, B) = ⌐A B

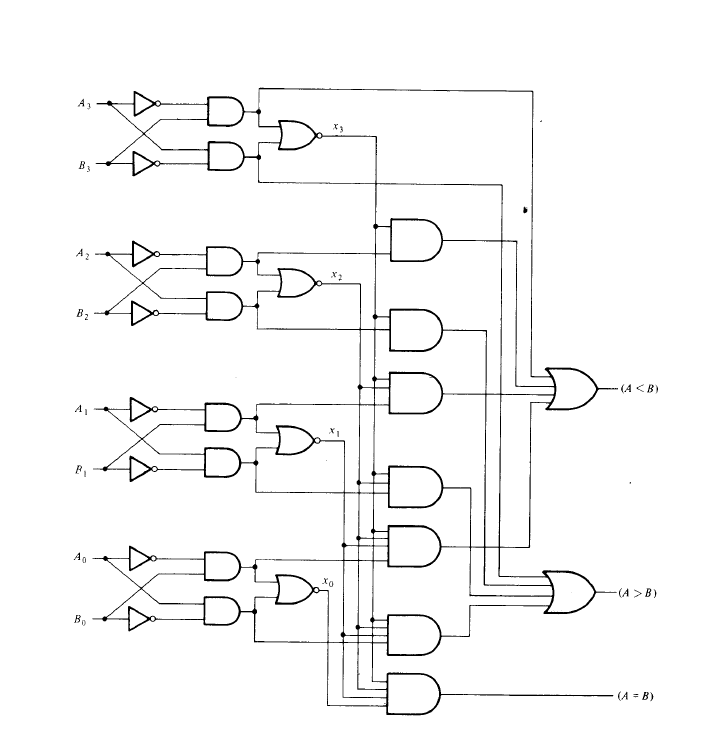
Y (A, B)= ⌐A⌐B + AB

Z (A, B)= ⌐AB

Procedemos a diseñar el circuito correspondiente:



Con el circuito anterior tenemos un comparador de dos bits, lo que equivale a replicar este circuito de manera que sea aplicado en palabras de 6 bits, es decir replicar el circuito 6 veces.



1

En este caso el ejemplo es de un comparador de 4 bits, debido a que el comparador de 6 bits es algo más extenso, pero la idea es básicamente replicar con el circuito base y así poder obtener resultados correctos. Además el circuito cuenta con una línea habilitadora que es la que da el permiso para realizar las operaciones, además de dos banderas, la SF, y la ZF, que de acuerdo con el resultado varían, de la siguiente manera:

|  |  |  |
| --- | --- | --- |
| Operación | Bandera ZF | Bandera SF |
| A = B | 1 | 0 |
| A > B | 0 | 1 |
| A < B | 0 | 0 |

Tabla2. Combinación utilizada por la familia de procesadores x86

**Bit Scan Forward (BSF)**

Este circuito se encarga de comprobar el primer índice donde se encuentre un 1 en una palabra de 6 bits, por lo tanto el índice debe estar dado en al menos 3 bits.

Para diseñar este circuito, tomamos en cuenta una pequeña tabla de verdad:

|  |  |
| --- | --- |
| Entrada | Salida |
| 000000 | 000 |
| 000001 | 001 |
| 00001X | 010 |
| 0001XX | 011 |
| 001XXX | 100 |
| 01XXXX | 101 |
| 1XXXXX | 110 |

Lo que quisimos conseguir con esta tabla, fue disminuir considerablemente la cantidad de bits de entrada que debíamos comprobar y la cantidad de casos para cada salida posible en el índice. Utilizando los términos “no importa” podemos ver que la cantidad de entradas que necesitamos es cada vez menor conforme el índice aumenta, ya que con sólo encontrar un uno en la posición en la que estamos, no es necesario comprobar los bits siguientes. A partir de esta idea concebimos un circuito que considere únicamente las entradas que se necesitan.

Una vez más, se necesita una bandera de comprobación de cero, la cual devuelve 0 si todos los bits de la entrada son 0, y un 1 en el caso contrario. Para este objetivo se utilizó la misma técnica de los circuitos anteriores, la cual no es un reto muy difícil como para mencionarlo aquí.

Conclusiones

**Yáser Castillo Bravo.**

* Uno de los objetivos generales de este curso y detallados en el programa de éste, es el de aprender y aplicar los elementos básicos del diseño digital. Como estudiante de este curso y participante de la realización de este proyecto siento que una evaluación de éstas le permite al estudiante ir más allá del papel en el cual muchas veces se realizan las pruebas cortas y tareas, con un concepto diferente y con una visión diferente pero sin perder el enfoque de los temas, este proyecto me ayuda a entender realmente cómo se trabajan los circuitos combinatorios en un dispositivo determinado, en este caso una ALU. Por otro lado me sirve ya que ésta es un área en la que más adelante y con el pasar del tiempo como estudiante tengo que desarrollar, para aprobar cursos siguientes, y a la larga trabajar en algún área relacionada con este tipo de implementación, o que sea necesario de conocimientos de este tipo.

**L. Alonso Vega Brenes**

* Debo decir que este tipo de proyectos me apasionan, ya que me dejan salir de las típicas tareas programadas y de las cansadas tareas de cálculo. Considero que el poder diseñar algo como un circuito, y poder comprobar su correcto funcionamiento es una experiencia placentera, que deja la sensación de haber hecho un buen trabajo.
* Creo que los circuitos digitales, a pesar de la incertidumbre de que tenga que trabajar algún día con ellos, me permiten tener una idea muy clara y sencilla de cómo es que funcionan estas máquinas que llamamos computadoras, inclusive las poderosas supercomputadoras. Todo es algo básico que se puede construir con pequeños bloques.
* Sinceramente pienso que tener este tipo de conocimientos, aunque sea desvalorizado por algunos, es importante para tener una idea de cómo funcionan los dispositivos que utilizamos día a día.

Anexos

## Plan de pruebas

**Sumario de los circuitos:**

* **Sumador-Restador**
  + Sumas
    - Con números grandes.
    - Con números que difieren en el signo.
  + Restas
    - De negativos.
    - De números grandes.
    - Con signos de entrada distintos.
  + Modificar la señal de acarreo entrante.
* **Multiplicador**
  + Números grandes
  + Signos
    - Distintos
    - Idénticos
* **Lógico**
  + Probar distintos valores de entrada.
  + Cambiar el índice de función.
* **Comparador**
  + Modificar las entradas para ir cambiando el mayor valor.
  + Consultar la tabla para verificar los resultados correctos.
* **BSF**
  + Modificar la palabra de entrada.
  + Comprobar que el índice sea el apropiado.

## Tablas e índices

|  |  |
| --- | --- |
| Operación | Índice |
| AND | **000** |
| OR | **001** |
| XOR | **010** |
| NOT | **011** |
| ShL | **100** |
| ShRL | **101** |
| ShRA | **110** |

**Índice de operaciones para el circuito lógico**

|  |  |  |
| --- | --- | --- |
| Comparación | Bandera ZF | Bandera SF |
| A = B | **1** | **0** |
| A < B | **0** | **1** |
| A > B | **0** | **0** |

**Banderas de salida para cada caso del circuito comparador**

Referencias Bibliográficas

1. Mano Morris. Arquitectura de Computadoras.
2. Presentaciones de diapositivas vistas en clase.