USO DEL LENGUAJE ENSAMBLADOR PARA PRÁCTICAS DE Aritmética

Julio Sahuquillo[1], Juan Carlos Cano[1], José Flich[1] y Salvador Petit[1].

[1]*Departamento de Informática de Sistemas y Computadores*
E. U. I.–Universidad Politécnica de Valencia–
e_mail: { jsahuqui, jucano, jflich, spetit} @ disca.upv.es

RESUMEN: La Unidad Aritmético Lógica (ALU) junto con la unidad de control (UC), son las unidades funcionales más relevantes del procesador.

El estudio de la UC es con diferencia el más abordado por los autores del área de conocimiento, desplazando a un segundo plano el estudio de la aritmética ha sido siempre un tema marginal. Lo cual queda patente tanto en clases magistrales como en las prácticas de laboratorio.

Este trabajo presenta un nuevo enfoque en las prácticas de laboratorio de la ALU, enfatizando el aspecto práctico sobre el teórico de cara a una formación más sólida del futuro Ingeniero en Informática.

1.- INTRODUCCIÓN.

La aritmética del computador se estudia en las asignaturas de Estructuras de Computador, dentro de los planes de estudio de Ingeniería Informática e Ingeniería Técnica en Informática. El hecho de que el tema se encuentre ligado con la lógica digital, ha provocado que las prácticas de laboratorio se realicen (en caso de que se realicen) mediante simuladores de circuitos digitales.

En líneas generales, las prácticas consisten en la implementación de circuitos aritméticos (sumadores y restadores). En opinión de los autores, este tipo de prácticas poco aportan a la formación del futuro ingeniero, debido a que los conceptos teóricos ya se han estudiado en clase, y en el laboratorio los alumnos simplemente implementan, utilizando el simulador, el mismo circuito (mismos componentes y conexiones) que el estudiado en clases magistrales.

El principal concepto en los temas de sumadores es la composición de circuitos. Para esto, se utilizan circuitos pequeños para la composición de circuitos más grandes con análogo funcionamiento. Por ejemplo, la realización de un circuito sumador convencional o CPA (siglas de Carry Propagated Adder) de 16 bits a partir de CPAs menores (de 8 ó 4 bits). La comprensión del funcionamiento de dichos circuitos es relativamente sencilla, y fácilmente asimilable por el alumno en clases magistrales; aún así, en muchas universidades se insiste en realizar prácticas de laboratorio, desde nuestro punto de vista poco o nada pragmáticas.

En este trabajo se propone utilizar el lenguaje ensamblador de cara a realizar y poner en práctica la composición de circuitos. Por tanto, se presenta un enfoque práctico para acercar al alumno hacía esta temática amen de formarle de cara al futuro ejercicio profesional. Debido a que en nuestra Universidad los alumnos estudian el MIPS R2000 en el primer cuatrimestre, en este trabajo se utiliza el citado lenguaje de cara a abordar la implementación práctica.

2. PRESENTACIÓN DEL PROBLEMA: EJEMPLO

Como ejemplo del enfoque propuesto, en este trabajo se aborda la implementación mediante lenguaje ensamblador de la realización de operaciones aditivas de dos números de 64 bits (o múltiplos de 32) utilizando la ALU de un procesador real de 32 bits.

Las CPUs tradicionalmente proporcionaban el valor de los indicadores de resultado en un registro especial. Estos indicadores se han estudiado en clase. Se pide, como ejercicio complementario que el alumno obtenga los indicadores de Z(Zero), N (Negative), C(Carry) y O(Overflow), mediante el lenguaje ensamblador.

El alumno debería intentar obtener él solo la solución que pasa por sumar independientemente la parte baja (bits del 31 al 0) de la parte alta (bits del 63 al 32). Como se aprecia en la Figura 1, se necesita tener en cuenta el acarreo del bit de mayor peso de la parte baja para sumárselo al resultado de la suma de la parte alta.

Aunque este estudio se realiza mediante el MIPS R2000, el ejercicio se puede particularizar para cualquier lenguaje ensamblador conocido por el alumno.

Solución tradicional

La Figura 1 muestra el diagrama de bloques de un CPA de 64 bits implementado a partir de dos CPAs de 32 bits conectados en serie. Como se aprecia el acarreo saliente (C_{31}) del CPA que suma los 32 bits de menor peso se conecta con la entrada de acarreo del CPA que suma los bits de mayor peso.

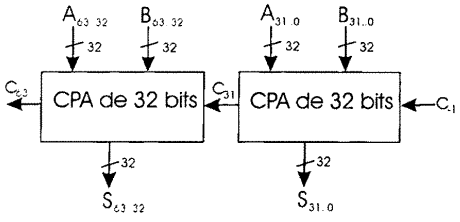


Figura 1. Diagrama de bloques de un CPA de 64 bits a partir de CPAs de 32 bits.

El esquema es sencillo y fácil de entender por lo que creemos que para su comprensión no es necesario montar el circuito en el laboratorio ya sea físicamente o mediante un simulador.

Solución propuesta: objetivos perseguidos

Teniendo en mente el problema descrito, el ejercicio persigue que se implemente la solución mediante el uso del lenguaje ensamblador con miras a que el alumno:

1. Madure en el aspecto práctico, utilizando la funcionalidad de la ALU para realizar operaciones con números más grandes (mayor número de bits) que los circuitos que en ella se integran.

2. Perciba y aprecie la importancia del lenguaje ensamblador, ya que minimiza la distancia entre la teoría y práctica.

Que duda cabe, que los aspectos citados avivan y despiertan, más si cabe, el interés del alumno por la materia.

3. DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

El problema o práctica a realizar se basa en implementar un programa que nos permita sumar dos números A y B de 64 bits, con una ALU de sólo 32, teniendo en cuenta que:

- Los datos a sumar se almacenarán inicialmente en memoria a partir de la dirección 0x10000000.
- Se supone que primero se almacena el dato A y a continuación el dato B.
- Se desea almacenar el resultado de la operación aritmética a continuación del dato B.

4. RESOLUCIÓN

Definición de los datos en memoria y cargado en registros

La ubicación de los datos en memoria debe ser el mostrado en la

Tabla 1.

Dirección	Contenido
0x1000 0000	$A_{31..0}$
0x1000 0004	$A_{63..32}$
0x1000 0008	$B_{31..0}$
0x1000 000C	$B_{63..32}$

Tabla 1. Ubicación de los datos en memoria.

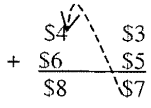
Para esto, las directivas de datos asociadas son:

```
    .data 0x10000000
A:      .word A31..0, A63..32     # Reservamos espacio para el sumando A
B:      .word B31..0, B63..32     # Reservamos espacio para el sumando B
R:      .space 8                  # Reservamos espacio para el resultado
```

Para poder operar con la ALU, hay que cargar los datos en los registros. Colocaremos la parte baja de A en $3 y en $4 la parte alta. B lo almacenaremos de forma que su parte baja esté en $5 y la parte alta en $6. Por último, el resultado lo obtendremos en $7 y $8 (parte baja y alta respectivamente). Una posible solución es:

```
__start:  la $4,A          # Cargamos la dirección de A para poder
          lw $3, 0($4)     # cargar primero la parte baja y luego
          lw $4, 4($4)     # cargar la parte alta.
          la $6,B          # Hacemos lo mismo con el segundo sumando
          lw $5, 0($6)
          lw $6, 4($6)
```

Esquemáticamente, se desea realizar:



Obtención de la suma

La suma de dos dobles palabras (números de 64 bits) se puede dividir en cuatro partes. En la primera, se sumarían las dos palabras de menor peso. A continuación se calcularía el acarreo de esta suma. Luego sólo nos queda sumarle este acarreo a la parte alta de A para finalizar sumando este resultado a la parte alta de B. Veamos el código:

```
add $7,$3,$5      # Hacemos la suma de las palabras de menos peso
sltu $8,$7,$3     # Ponemos en $8 el acarreo
                  # Hay acarreo si el resultado es menor que un sumando
add $8,$8,$4      # Sumamos la parte alta de A con el acarreo
add $8,$8,$6      # Sumamos el resultado anterior con la parte alta de B,
```

Obtención de los indicadores de resultado

Indicador de Cero (Z)

El indicador de cero se activa si el resultado de la operación es cero.

```
or $9,$8,$7    # En $9 tendremos distinto de 0 si hay algún 1 en la parte
               # alta del resultado $8 o en la parte baja $7
```

```
sltiu $9,$9,1   # Ponemos $9 a 1 si el resultado anterior es menor que 1
                # (es igual a 0). En caso contrario, pondremos un 0.
                # Por tanto $9=Z
```

Indicador de signo (N)

Este indicador se utiliza cuando se trabaja con números con signo, y se activa cuando el resultado es negativo. El valor del indicador es igual al valor del bit más significativo del resultado.

```
srl $9,$8,31    # Desplazamos el registro que contiene la palabra de mayor peso
                # 31 posición para chequear el valor de su bit de mayor peso: 63
                # El bit correspondiente es el de menor peso del registro $9
                # Por tanto $9=N
```

Indicador de Desbordamiento (Overflow)

El desbordamiento nos indica que el resultado de una operación necesita más bits de los que se dispone para almacenarlo. Internamente, se implementa con la XOR del bit, de acarreo del último y penúltimo bit. Sin embargo se puede calcular de otra forma (en una operación aditiva): si sumamos dos números de distinto signo no puede haber desbordamiento, sino que sólo se produce desbordamiento cuando se suman dos números positivos y se obtiene uno negativo o cuando se suman dos negativos y se obtiene un resultado positivo. Por tanto, calcularemos el indicador para sumas de números de igual signo.

```
xor $9,$4,$6    # En el bit más significativo del resultado obtenemos un 1 si los
                # sumandos tienen distinto signo y 0 si tienen el mismo signo.
srl $9,$9,31    # Colocamos el bit que nos interesa en la posición de menos peso
bne $9,$0,fin   # Si tienen signos distintos, $9=1 (no hay desbordamiento)
xor $9,$8,$6    # Si tienen el mismo signo verificamos si el resultado tiene
                # el mismo signo que los sumandos ($9=1)
srl $9,$9,31    #Colocamos el bit que nos interesa en el lugar de menos peso
fin:            # $9=O
```

Indicador de acarreo (C)

Este indicador se activa cuando al sumar los dos bits de mayor peso se genera acarreo ("nos llevamos uno"). La interpretación de su valor varía en función de si se trabaja con números con signo o números sin signo.
Cuando se trabaja con números sin signo un valor igual a uno indica que se ha producido desbordamiento.
Al realizar la suma de números sin signo se activa cuando el resultado sea menor que cualquiera de los sumandos, el código correspondiente es:

```
sltu $9,$8,$6
bne $9,$0,fin   # Hay acarreo ($9=1) si el resultado es menor que un sumando
bne $8,$6, fin  # Si el resultado es mayor que un sumando (salta) no
                #hay acarreo ($9=0)
sltu $9,$7,$5   #AQUÍ LLEGAMOS SI SON IGUALES
                # se comprueba las partes menos significativas y si
                # la del resultado es menor, habrá acarreo. En caso contrario, no.
fin:            # C=$9
```

Programa completo

Con todo lo expuesto anteriormente, el programa completo resultaría:

```
        .data 0x10000000
A:      .word A31..0, A63..32   # Reservamos espacio para el sumando A
B:      .word B31..0, B63..32   # Reservamos espacio para el sumando B
R:      .space 8                # Reservamos espacio para el resultado
        .text
__start: la $4,A                # Cargamos la dirección de A para poder
        lw $3, 0($4)            # cargar primero la parte baja y luego
        lw $4, 4($4)            # cargar la parte alta.
        la $6,B                 # Hacemos lo mismo con el segundo sumando
        lw $5, 0($6)
        lw $6, 4($6)
        add $7,$3,$5            # Hacemos la suma de las palabras de menor peso
        sltu $8,$7,$3           # Hay acarreo si el resultado es < que un sumando
        add $8,$8,$4            # Sumamos la parte alta de A con el acarreo
        add $8,$8,$6            # Sumamos también la parte alta de B.
        la $10, R               # Cargamos la dirección del resultado para poder
        sw $7, 0($10)           # almacenar primero la parte baja y
        sw $8, 4($10)           # en la dirección siguiente la parte alta del resultado
```

Generalización para k>2

Hasta ahora se ha visto cómo sumar dos números de 64 bits (2 palabras de 32 bits) con un procesador que sólo puede trabajar con 32 bits. Sin embargo, lo estudiado se puede extender para cualquier numero de k*32 bits. A continuación, se presenta el código para sumar dos números para cualquier valor de k. Con las siguiente restricciones:

Los números a sumar se almacenan en A y B en formato *little endian*, separando cada palabra por comas. Ej: A: .word 5678,1234.

En la línea (2) debemos sustituir 4*k por el resultado de multiplicar 4 por el número de bytes que ocupan los sumandos. Ej: .space 8.

En la línea (3) tenemos que cambiar k por el número de palabras (de 32 bits) de los sumandos. Ej: K: .word 2.

Cada registro se utiliza con los fines especificados en la Tabla2.

Registro	Contenido
$3	Dirección de la palabra de A que estamos operando.
$4	Dirección de la palabra de B que estamos operando.
$5	Dirección de la palabra donde hay que almacenar el resultado de la palabra en curso.
$6	Número de palabras que nos quedan por operar.
$7	Registro temporal donde llevamos la suma de la palabra en curso y el acarreo.
$8	Registro temporal usado para leer de memoria de los sumandos.

Tabla 2. Uso de los registros.

```
# Suma.S
# Calcula la suma de dos números de k*32 bits
        .data
  A:    .word A1,A2...Ak # Reservamos memoria para el primer sumando
  B:    .word B1,B2...Bk # Reservamos memoria para el segundo sumando
  R:    .space 4*k       # (2) Reservamos memoria para el resultado
  K:    .word k          # (3) Le indicamos al programa el tamaño (en palabras)
                         # de cada número
        .text
__start :la  $3,A        # Cargamos el puntero al primer sumando
        la  $4,B         # Cargamos el puntero al segundo sumando
        la  $5,R         # Cargamos el puntero al resultado
        la  $6,K
        lw  $6,0 ($6)    # Cargamos el numero de palabras que hay que operar
        or  $7,$0,$0     # Ponemos a cero el acarreo, puesto que en la primera
                         # suma no nos llevamos nada de la suma anterior.
bucle:  lw  $8,0 ($3)    # Leemos la palabra a operar del primer sumando
        add  $7,$7,$8    # Sumamos la palabra del primer sumando al acarreo
                         # anterior
        lw  $8,0 ($4)    # Cargamos la palabra del segundo sumando
        add  $7,$7,$8    # Sumamos el segundo sumando a la suma parcial
                         # anterior
        sw  $7,0 ($5)    # Almacenamos el resultado obtenido en memoria
        sltu $7,$7,$8    # Calculamos el acarreo para la siguiente palabra
        addi $3,$3,4     # Apuntamos a la siguiente
        addi $4,$4,4     # palabra a operar tanto de los dos sumandos como del
        addi $5,$5,4     # resultado.
        addi $6,$6,-1    # Ahora ya nos queda una palabra menos por operar
        bne  $6,$0,bucle # Si aún quedan palabras por sumar, volvemos para
                         # sumar la siguiente palabra.
```

5.- CONCLUSIONES

En este trabajo se ha presentado un nuevo enfoque de realizar las prácticas de circuitos aritméticos, considerando el lenguaje ensamblador como herramienta.

En opinión de los autores este nuevo enfoque motivará a los alumnos no sólo en el estudio del lenguaje ensamblador sino a consolidar sus conocimientos de arquitectura de computadores. Además este acercamiento entre la teoría y la práctica, incentiva al alumno en la utilización de este lenguaje. Por último señalar, que la mayor parte de los alumnos disponen de PCs lo que les habilita para continuar el trabajo en sus casas y potenciar, más si cabe, el "aprender haciendo".