CONSIDERACIONES DEL TEMA DE PROGRAMACIÓN EN ENSAMBLADOR

Este documento se ha creado para facilitar al alumno el estudio autónomo dada la situación excepcional que padecemos. Como punto de partida se asume que el alumno tienes los conocimientos adquiridos del tema de "Instrucciones y direccionamientos", por lo que las referencias a este tema serán continuas. La clase de inicio del tema muestra el juego de instrucciones del computador 88110, las pseudoinstrucciones del programa ensamblador y el primer ejemplo de las transparencias. A continuación se muestran instrucciones y fragmentos de código que el alumno sabe cómo resolver en el estándar IEEE 694, pero que no es trivial una vez visto el juego de instrucciones del 88110.

CARGA DE UN DATO DE TAMAÑO INFERIOR A UNA PALABRA

Este problema se plantea cuando deseamos inicializar un registro con un valor pequeño para utilizarlo, por ejemplo, como índice de un contador. En el estándar IEEE se realizaría con la instrucción LD, pero en el 88110 la instrucción ld siempre realiza un acceso a memoria. Por ejemplo, si necesitamos inicializar el registro r3 con el valor 5, en el estándar se ejecutaría la instrucción

```
LD .R3,#5
```

En el caso del 88110 no se puede utilizar ld y se usará una instrucción aritmética o lógica:

```
or r3,r0,5
```

```
add r3,r0,5
```

La primera alternativa realiza una operación lógica bit a bit de la parte baja del registro r0 (que siempre contiene el valor 0) con el valor inmediato 5. El resultado son 16 bits que contienen el valor 5 y que se almacenan en los 16 bits menos significativos del resultado (r3). Los 16 bits más significativos del resultado se almacenan los 16 bits más significativos de r0 (a 0).

La segunda alternativa suma los 32 bits de r0 (0) con el valor inmediato 5 al que se le ha hecho una extensión de signo a 32 bits. En definitiva r3 se almacena con la suma 0+5 en 32 bits.

CARGA DE UN DATO INMEDIATO DE UNA PALABRA

La instrucción en el estándar es: **LD .R1, #H'12345678** ; R1 <- H'12345678

Esta instrucción carga el dato de 32 bits (que supondremos que es el tamaño de palabra) expresado en hexadecimal 12345678 en el registro R1. Puesto que el dato inmediato ya ocupa una palabra, esta instrucción debería ocupar dos palabras en el computador. La instrucción ld del 88110 no es utilizable para su equivalente de la instrucción propuesta, porque siempre realiza un acceso a memoria.

Como es sabido, todas las instrucciones del 88110 ocupan una palabra de 32 bits y por tanto los datos necesarios para codificar una instrucción no "caben" en una palabra. Para ello haremos uso de la instrucción or y tendremos en cuenta que el registro r0 siempre contiene el valor 0. El resultado es la secuencia de dos instrucciones:

```
or   r1,r0,0x5678
or.u r1,r1,0x1234
```

La primera instrucción realiza una operación lógica or bit a bit de los 16 bits menos significativos del r0 con el dato inmediato de 16 bits. El resultado se almacena en los 16 bits menos significativos de r1 y los 16 bits más significativos de r0 se copian en r1. Puesto que el resultado de una operación lógica or de un 0 con cualquier otro dato es el otro dato, el resultado de la primera instrucción es que r1 contiene el dato de 32 bits 0x00005678.

A continuación se ejecuta la instrucción or.u que realiza la operación lógica or de los 16 bits más significativos de r1 (16 bits a 0) con el dato inmediato de 16 bits. Además, se copian los 16 bits menos significativos de r1 a r1, es decir, los 16 bits menos significativos de r1 no se modifican y que son los que inicializamos en la instrucción anterior. El resultado es que r1 pasa a contener 0x12345678.

CARGA UNA PALABRA ALMACENADA EN UNA POSICIÓN DE MEMORIA

La instrucción en el estándar es: **LD .R1, /H'98765432** ; R1 <- m(H'98765432)

La instrucción carga en R1 el contenido de la posición de memoria 0x98765432. Si esta dirección de memoria contiene el valor 0xABCDEF01 almacenará este valor en R1. Al igual que en el caso anterior, la dirección involucrada en esta instrucción ocupa una palabra. Esta instrucción no existe como tal en el 88110, puesto que el direccionamiento directo a memoria no está entre los proporcionados por el computador. El único direccionamiento a memoria disponible para acceso a datos es el direccionamiento relativo a registro base.

Al igual que en el caso anterior los datos de la instrucción no "caben" en una palabra. Para obtener el equivalente del 88110 de esta instrucción, cargaremos la dirección de memoria a la que deseamos acceder (0x98765432) en un registro y a continuación accederemos a memoria mediante direccionamiento relativo a registro base utilizando como registro base el registro cargado anteriormente y desplazamiento 0. La secuencia equivalente sería:

```
or    r1,r0,0x5432
or.u  r1,r1,0x9876
ld    r1,r1,r0
```

Las dos primeras instrucciones de la secuencia obtiene como resultado la dirección efectiva del direccionamiento cargada en el registro r1: 0x98765432. La tercera instrucción utiliza el registro r1, cargado en las dos instrucciones anteriores, como registro base en una acceso de lectura en memoria. El desplazamiento utilizado es el registro r0, que siempre contiene el valor 0. Por tanto la dirección efectiva del direccionamiento relativo a registro base es (r1+r0) 0x98765432 y la tercera instrucción carga el contenido de esta dirección en el registro r1.

CARGA DE LA DIRECCIÓN ASOCIADA A UNA ETIQUETA

La utilización del programa ensamblador permite la simplificación del uso de direcciones de memoria de forma absoluta y de desplazamientos en las instrucciones de salto. Un buffer de memoria o vector de datos puede declararse en el estándar IEEE694 y en el ensamblador del 88110 mediante la pseudoinstrucciones:

```
ORG 0x400

VECTOR: DATA 1,2,3,4

VECTOR2: RES 4
```

De esta forma sabemos que VECTOR es una etiqueta que representa la dirección de comienzo del vector (1,2,3,4) cuyo valor es 0x400. VECTOR2 es una etiqueta asociada a la dirección 0x410, puesto que cada uno de los datos de VECTOR es de 4 bytes y el direccionamiento de la memoria es a nivel de byte. La forma de cargar la dirección de comienzo de VECTOR en el registro 20 del 88110 sería:

```
or r20,r0,0x400
```

En este caso no estamos utilizando la etiqueta y el problema surge cuando movemos la dirección de memoria en la que está almacenado el vector. Si utilizáramos la etiiqueta esto no supondría problema, puesto que la etiqueta estaría asociada a la nueva dirección tras el nuevo ensamblado. Lo que se muestra es la forma de cargar la dirección asociada a la etiqueta VECTOR en el registro r20 (transparencia 24).

```
or r20,r0,low(VECTOR)
or.u r20,r20,high(VECTOR)
```

La primera instrucción hace el or bit a bit de los 16 bits menos significativos de r0 (0) con los 16 bits menos significativos asociados a la etiqueta VECTOR en el ejemplo 0x400). El resultado (0x400) se almacena en los 16 bits menos significativos de r20 y los 16 bits más significativos de r20 se copian con los 16 bits más significativos de r0 (0). La segunda instrucción hace or bit a bit del los 16 bits más significativos de r20 (que se han puesto a 0 en la instrucción anterior) con los 16 bits más significativos de la dirección asociada a VECTOR (0) y se copian los 16 bits

menos significativos de r20 en los 16 bits menos significativos de r20. Como resultado r20 acaba conteniendo el valor 0x400 que es la dirección asociada a VECTOR.

SALTOS CONDICIONALES

La ejecución de saltos condicionales en el estándar se apoya en la existencia de los flags de estado del computador. Son condiciones que la ALU actualiza en las instrucciones aritméticas, lógicas y de comparación acerca del resultado que se ha producido en la ALU. Por ejemplo si deseamos comprobar si el contenido de los registros R1 y R2 son iguales y saltar a un fragmento de código que comienza en la etiqueta IGUALES se realizaría

```
CMP R1,R2
BZ IGUALES
; código de distintos
```

Puesto que CMP realiza una resta que no almacena el resultado en ningún sitio, pero sí actualiza los flags de estado, el flag Z estaría activo cuando R1 y R2 son iguales (su resta es cero) y por tanto el salto de la instrucción BZ sería efectivo. El código que sigue a esta instrucción se ejecutaría cuando no se toma el salto, es decir, cuando R1 no es igual a R2.

El 88110 no tiene dichos flags de estado y la condición de salto hay que generarla mediante la instrucción cmp y a continuación comprobar la condición generada mediante la instrucción de salto condicional bb0 o bb1. La instrucción cmp genera 14 condiciones diferentes en el registro destino de la instrucción relacionadas con los registros operandos de la misma (últimos registros). A modo de ejemplo se muestra el código equivalente al código anterior:

```
cmp r7,r1,r2
bb1 2, r7, IGUALES
; código de distintos
```

En la primera instrucción se han generado las 14 condiciones en el registro r7 (transparencia 18). La condición que está almacenada en el bit número 2 se pone a 1 cuando r1 y r2 son iguales. La siguiente instrucción (bb1) es una instrucción de salto condicional que toma el salto (a la instrucción asociada a la etiqueta IGUALES) cuando el bit 2 de r7 es 1. Por tanto la instrucción bb1 tomará el salto cuando r1 sea igual a r2.

Obsérvese que aunque el juego de instrucciones del 888110 exige establecer un desplazamiento con signo como dato de la instrucción bb1, el ensamblador del 88110 permite indicar una etiqueta y el programa ensamblador calcula el desplazamiento que hay que almacenar en la instrucción en el binario ejecutable.

Adicionalmente, el código que se muestra a continuación es equivalente al anterior

```
cmp r7,r1,r2
bb0 3, r7, IGUALES
; código de distintos
```

EJEMPLO: CÓDIGO DE UN BUCLE SENCILLO

Para ilustrar todos los conceptos aclarados en este documento se propone realizar un programa que sume todos los números enteros hasta un límite que está contenido en una posición de memoria etiquetada con NUMERO. A continuación se muestra el código:

```
NUMERO: DATA 5
      or r20,r0,low(NUMERO)
      or.u r20,r20,high(NUMERO) ; r20 se carga con la dir.
                                ; asociada a NUMERO
      ld r2,r20,r0 ; r2 se carga con el contenido de NUMERO (5)
      or r3,r0,r0  ; r3 se inicializa con un contador
      or r4,r0,r0  ; r4 se inicializa con el resultado de la suma
buc:  cmp r7,r3,r2 ; Si el contador es igual a NUMERO
      bb1 eq,r7,fin; Se salta a fin y se acaba
      addu r4,r4,r3; Se acumula el contador a la suma
      addu r3,r3,1 ; Se incrementa el contador
      br buc       ; Salto incondicional al bucle
fin:  stop
```

La resolución del problema se basa en inicializar un registro en el que iremos acumulando la suma de cada uno de los índices del contador. Se inicializará un contador a 0 que se irá a acumulando a la suma en cada iteración del bucle. El finalizar la iteración del bucle se incrementa el contador. El bucle finaliza cuando el contador alcanza (es igual) al valor contenido en la etiqueta NUMERO, que en la fase de inicialización se ha cargado en r2.