DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Sistemas Digitales

Preparación parcial-Manipulación de bits

Primer Cuatrimestre 2024

Sistemas Digitales
DC - UBA

Introducción

Preparación para el parcial

Este material tiene:

- Esquema general del parcial.

Preparación para el parcial

Este material tiene:

- Esquema general del parcial.
- Ejemplo de ejercicio de manipulación de bits y técnicas comunes.

Esquema general

Sobre el parcial

El parcial supone un buen entendimiento de temas de representación numérica, operaciones aritmético-lógicas y de circuitos combinatorios y secuenciales. Su foco va a estar en el uso de la arquitectura `RISC V` a través de su lenguaje ensamblador. En particular el uso de instrucciones del módulo base (RISC V-I) con algunas excepciones, como pueden ser las instrucciones de multiplicación y división, según cada caso.

Manipulación de bits

Manipulación de bits

A la hora de almacenar datos y procesarlos, muchas veces tenemos que diseñar la forma en la que vamos a modificar un único dato, para esto vamos a presentar una serie de técnicas comunes que se basan en:

- Definir constantes específicas (máscaras).
- Aplicar **and, or, desplazamientos**.
- Leer y escribir de/a memoria.

Nota sobre constantes

Cuando definimos constantes en la sección de datos (`.data`) utilizando las instrucciones de ensamblado `.byte` y `.word` los elementos que definimos se van a cargar desde la posición más baja de memoria en adelante, entonces en el siguiente ejemplo las dos constantes son equivalentes.

.byte y .word

En RISC V sería:

```
1 .data
2 # las constantes se definen desde el byte mas bajo
3 # mascara: byte0 byte1 byte2 byte3
4 mascara: .byte 0xDD 0xCC 0xBB 0xAA
5 # estas constantes son iguales
6 mascara2:.word 0xAABBCCDD
```

Limpiar un dato

Manipulación de bits

Es necesario conocer cómo están estructurados los datos para operar con ellos, a la forma en la que están almacenados la llamamos **empaquetamiento**(packing). Por ejemplo, un pixel puede estar representado como cuatro bytes que representan transparencia, valor de rojo, valor de verde y valor de azul.

Bits	31-24	23-16	15-8	7-0
s0	alpha	rojo	verde	azul

Manipulación de bits

Si quisiéramos, por ejemplo preservar solamente el componente rojo y la transparencia podríamos utilizar una constante inicializada apropiadamente y un **and** para conseguirlo.

Bits	31-24	23-16	15-8	7-0
s0	alpha	rojo	verde	azul
s1	0xFF	0xFF	0x00	0x00
and s0, s0, s1	alpha	rojo	0x00	0x00

Mantener alpha y rojo

En RISC V sería:

```
1  .data
2  # las constantes se definen desde el byte mas bajo
3  # mascara: byte0 byte1 byte2 byte3
4  mascara: .byte 0x00 0x00 0xFF 0xFF
5  # lo mismo sucede con el dato, si se imprime
6  # se leera: 0x357A4E3C
7  pixel: .byte 0x3C 0x4E 0x7A 0x35
8  .text
9  main:
10     #cargamos la mascara que
11     #limpia verde y azul
12     lw t0, mascara
13     #cargamos el pixel
14     lw t1, pixel
15     and a0, t0, t1
```

Negar un dato

Negando un dato

Si quisiéramos negar un dato podemos hacer un **xor** con una máscara de todos unos.

Bits	31-24	23-16	15-8	7-0
s0	11001111	11001100	00000000	11111111
s1	11111111	11111111	11111111	11111111
xor s0, s0, x1	00110000	00110011	11111111	00000000

Recordemos que el xor vale 1 si solamente uno de los operandos vale 1 y el otro 0.

Negando un dato

En `RISC V` podemos cargar la máscara de todos unos de dos maneras, o bien lo cargamos como constante o como inmediato de valor -1, que será codificado como todos unos en 12 bits y luego extendido con signo a 32.

Negando con constante

En RISC V sería:

```
1  .data
2  # las constantes se definen desde el byte mas bajo
3  mascara: .byte 0xFF 0xFF 0xFF 0xFF
4  dato: .byte 0xF0 0x0F 0xFF 0x00
5  .text
6  main:
7      #cargamos la mascara de todos unos
8      lw s0, mascara
9      #cargamos el dato
10     lw s1, dato
11     #negamos con un xor
12     xor s0, s0, s1
```

Negando con inmediato

En RISC V sería:

```
1  .data
2  # las constantes se definen desde el byte mas bajo
3  dato: .byte 0xF0 0x0F 0xFF 0x00
4  .text
5  main:
6      #cargamos el dato
7      lw s0, dato
8      #negamos con un xori e inmediato de 12 bits
9      #extendido en signo a 32 (todos unos)
10     xori s0, s0, -1
```

Desempaquetar un dato

Desempaquetar un dato

A menudo queremos extraer un dato de menor tamaño de un dato y operar con él, para eso es habitual o bien limpiar el dato o extender el signo del dato de menor tamaño que extrajimos. Vamos a trabajar con cuatro bytes empaquetados en una palabra (s0).

Bits	31-24	23-16	15-8	7-0
s0	dato 3	dato 2	dato 1	dato 0

Imaginemos que queremos sumar el dato 3 y el dato 2.

Desempaquetar un dato

Supongamos primero que son datos sin signo, vamos a extraer el dato 2 con una máscara (s1) y luego con un desplazamiento de dos bytes a derecha (16 bits).

Bits	31-24	23-16	15-8	7-0
s0	dato 3	dato 2	dato 1	dato 0
s1(máscara)	0x00	0xFF	0x00	0x00
and t0, s0, s1	0x00	dato 2	0x00	0x00
srli t0, t0, 16	0x00	0x00	0x00	dato 2

Desempaquetar un dato

Para extraer el dato 3 con una máscara (s2) y luego con un desplazamiento de tres bytes a derecha (24 bits).

Bits	31-24	23-16	15-8	7-0
s0	dato 3	dato 2	dato 1	dato 0
s2(máscara)	0xFF	0x00	0x00	0x00
and t1, s0, s2	dato 3	0x00	0x00	0x00
srli t1, t1, 24	0x00	0x00	0x00	dato 3

Desempaquetar un dato

Ahora podemos sumar, hay que tener en cuenta que el resultado de la suma puede ocupar dos bytes.

Bits	31-24	23-16	15-8	7-0
t0	0x00	0x00	0x00	dato 2
t1	0x00	0x00	0x00	dato 3
add a0, t0, t1	0x00	0x00	suma 2	suma 1

Desempaquetar un dato

En RISC V sería:

```
1          .data
2          mascara1: .byte 0x00 0x00 0xFF 0x00
3          mascara2: .byte 0x00 0x00 0x00 0xFF
4          dato: .byte 0xF0 0x0F 0xFE 0x3C
5          .text
6          main:
7          #cargamos el dato
8          lw s0, dato
9          #cargamos las mascara
10         lw s1, mascara1
11         lw s2, mascara2
12         #limpiamos y sumamos
13         and t0, s0, s1
14         srli t0, t0, 16
15         and t1, s0, s2
16         srli t1, t1, 24
17         add a0, t0, t1
```

Desempaquetar un dato

Otra forma de desempaquetar un dato, sin usar máscaras es con desplazamientos, básicamente podemos desplazar el dato (en nuestro caso el byte) a izquierda como para borrar los datos a izquierda y luego a derecha, para completar con ceros.

Desempaquetar un dato

Para el dato 2 hagamos un desplazamiento de un byte (8 bits) a izquierda y luego tres bytes a derecha (24 bits). La idea es completar con ceros a derecha primero, y luego con ceros a izquierda hasta conseguir que quede solamente el byte que nos interesa.

Bits	31-24	23-16	15-8	7-0
s0	dato 3	dato 2	dato 1	dato 0
slli s0, s0, 8	dato 2	dato 1	dato 0	0x00
srli s0, s0, 24	0x00	0x00	0x00	dato 2

Desempaquetar un dato

Si el dato fuese en complemento a dos, o sea un byte empaquetado dentro de una palabra, podemos usar desplazamientos aritméticos para preservar el signo.

Desempaquetar un dato

Para el dato 2 hagamos un desplazamiento de un byte (8 bits) a izquierda y luego tres bytes a derecha (24 bits) con signo. La idea es completar con ceros a derecha primero, y luego con extensión de signo a izquierda hasta conseguir que quede solamente el byte que nos interesa.

Bits	31-24	23-16	15-8	7-0
s0	dato 3	dato 2	dato 1	dato 0
slli s0, s0, 8	dato 2	dato 1	dato 0	0x00
srai s0, s0, 24	signo	signo	signo	dato 2

Desempaquetar un dato

Para el dato 2 hagamos un desplazamiento de un byte (8 bits) a izquierda y luego tres bytes a derecha (24 bits) con signo. La idea es completar con ceros a derecha primero, y luego con extensión de signo a izquierda hasta conseguir que quede solamente el byte que nos interesa.

Bits	31-24	23-16	15-8	7-0
s0	0x0F	0xF0	0xC0	0x3E
slli s0, s0, 8	0xF0	0xC0	0x3E	0x00
srai s0, s0, 24	0xFF	0xFF	0xFF	0xF0

Empaquetar un dato

Empaquetar un dato

Si tuviésemos dos datos de un byte que queremos empaquetar en la parte alta de una palabra podemos hacer uso de una máscara o desplazamientos para limpiar la parte alta y luego dos desplazamientos y uso de `or`.

Empaquetar un dato

Queremos guardar los datos que están en s1 y s2, que son de un byte, en la parte alta de s0.Vamos a limpiar la parte alta de s0 primero.

Bits	31-24	23-16	15-8	7-0
s0	basura	basura	dato 1	dato 0
slli s0, s0, 16	dato 1	dato 0	0x00	0x00
srli s0, s0, 16	0x00	0x00	dato 1	dato 0

Empaquetar un dato

Ahora limpiamos y empaquetamos s1.

Bits	31-24	23-16	15-8	7-0
s1	basura	basura	basura	dato 2
slli s1, s1, 24	dato 2	0x00	0x00	0x00
srli s1, s1, 8	0x00	dato 2	0x00	0x00
or s0, s0, s1	0x00	dato 2	dato 1	dato 0

Empaquetar un dato

Ahora limpiamos y empaquetamos s2.

Bits	31-24	23-16	15-8	7-0
s2	basura	basura	basura	dato 3
slli s2, s2, 24	dato 3	0x00	0x00	0x00
or s0, s0, s2	dato 3	dato 2	dato 1	dato 0

Fin