Tutorial3.2

Shift instructions, Sign/Zero Extend Operations, Bitfield Operations

Lei Wang

lei.wang2@ucalgary.ca

UNIVERSITY OF CALGARY

signed integers and unsigned integers

- Unsigned integers
  - Range: 0 to $2^N - 1$, where N is number of bits
  - Eg: 8-bits register: ranges from 0 to 255
    - in binary, from 00000000 to 11111111
- Signed integers
  - Range: -2^{N-1} to $+2^{N-1} - 1$
    - 4-bits: -8 to +7
    - 8-bits: -128 to 127
    - 16-bits: -32768 to +32767
    - 32-bits: -2147483648 to +2147483647
    - 64-bits: -2^{63} to $+2^{63} - 1$

UNIVERSITY OF CALGARY

Signed integers

- Negating a number is done by:
  1. Taking the one's complement
     - Toggle all 0's to 1's, and vice versa
  2. Adding 1 to the result
  - Eg: find the bit pattern for -5 in a 4-bit register
    - +5 is 0101
    - One's complement: 1010
    - Add 1: 1011
  - Also works when negating negative numbers
    - Eg: -5 to +5
    - One's complement of 1011: 0100
    - Add 1: 0101
- All positive numbers will have a 0 in the left-most bit
  - and all negatives will have a 1
  - called the sign bit

UNIVERSITY OF CALGARY

Bitwise Shift Instructions

- Logical Shift Left
  - lsl Xd, Xn, Xm : 0 is shifted into rightmost bit
  - multiplication by a power of two
- Logical Shift Right
  - lsr Xd, Xn, Xm: 0 is shifted into leftmost bit
  - Division by a power of two, does not work for negative integers
- Arithmetic Shift Right
  - asr Xd, Xn, Xm
  - Sign bit is preserved, work for negative integers
  - eg: -8/2 = -4

```
mov  x20, -8          // 1111 … 1111 1000
mov  x21, 1           // 2^1 = 2
asr  x19, x20, x21    // 1111 … 1111 1100
```

Sign/Zero Extend Operations

- Signed Extend Byte
  - sxtb Wd, Wn
  - sign-extends bit 7 in Wn to bits 8-31
- Signed Extend Halfword
  - sxth Wd, Wn
  - sign-extends bit 15 in Wn to bits 16-31
- Signed Extend Word
  - sxtw Xd, Wn
  - sign-extends bit 31 to bits 32-64
- Unsigned Extend Byte/halfword/word
  - uxtb Wd, Wn
  - Zero-extend bits 8-31

sign-extend: use sign bit for extension (bit7, bit 15, bit 31)

zero-extend: use 0 for extension

Bitfield Operations

- Bitfield Insert
  - bfi *Wd, Wn, #lsb, #width*
  - source field occupies bits 0 to *(width - 1)* in *Wn*
  - bits *lsb* to *(lsb + width - 1)* in *Wd* are replaced by the source bitfield
- bitfield Extract and Insert Low
  - bfxil *Wd, Wn, #lsb, #width*
  - source bitfield is bits *lsb* to *(lsb + width - 1)* in *Wn*
  - bits 0 to *(width - 1)* in *Wd* are replaced by the source bitfield

Binary Multiplication

- **Multiplication is achieved by adding a list of shifted multiplicands according to the digits of the multiplier.**
- **Ex. (unsigned)**

```
   11                  1 0 1 1     multiplicand (4 bits)
X  13          X       1 1 0 1     multiplier  (4 bits)
--------       -----------------
   33                  1 0 1 1
  11                 0 0 0 0
________           1 0 1 1
  143            1 0 1 1
               -------------------
               1 0 0 0 1 1 1 1            Product (8 bits)
```

Coding practice
simple binary multipication

- assembly
- c

```c
#include <stdio.h>
int main()
{
  int multiplier, multiplicand, product
  multiplicand = 0b00001010;
  multiplier = 0b00001011;
  product = 0;
  printf("multiplier = 0x%02x (%d)
        multiplicand = 0x%02x (%d)\n\n",
        multiplier, multiplier,
        multiplicand, multiplicand);
  for (int i = 0; i < 4; i++) {
    if (multiplier & 0x1) {
      product = product + multiplicand;
    }
    multiplier = multiplier >> 1;
    multiplicand = multiplicand << 1;
  }

  printf("product = 0x%02x(%d)\n",
   product, product);
  return 0;
}
```

multiplicand: 1011(11)
multiplier: 1010(10)

```
    0000
   1011
  0000
 1011
```

product: 01101110(110)

```
lei.wang2@csa2:~/tutorial3$ ./ref
Multiplier = 0x0b (11) Multiplicand = 0x0a (10)

Product = 0x6e (110)
```

```
// Define the strings
define(multiplier, w19)
define(multiplicand, w20)
define(product, w21)
define(i, w22)

initialValues:      .string "Multiplier = 0x%02x (%d) Multiplicand = 0x%02x (%d) \n\n"
printProduct:       .string "Product = 0x%02x (%d)\n\n"

    .balign 4                       // Instructions word aligned
    .global main                        // Make "main" visible to the OS
main:                               // Main function, code starts
    stp x29, x30, [sp, -16]!            //
    mov x29, sp                     // Update FP to current SP (post-incr SP)

    mov multiplicand, 10                // Give multiplicand a value 00001010
    mov multiplier, 11                  // Give a multiplier a value 00001011
    mov product, 0                  // Give product a value 0
    mov i, 0                        // Give counter i a value 0

    adrp    x0, initialValues               // Set 1st arg of printf high
    add x0, x0, :lo12:initialValues         // Set 1st arg of printf low
    mov w1, multiplier                  // 2nd
    mov w2, multiplier                  // 3rd
    mov w3, multiplicand                // 4th
    mov w4, multiplicand                // 5th
        bl      printf                      // Print statement of innitial values

    b   test                        // Jump to test

forloop:                            // For loop
    tst multiplier, 0x1                 // test if bit-0 in multiplier == 1
    b.eq    nextIf
    add product, product, multiplicand          // add multiplicand to product once

nextIf:                             // If statement
    lsr multiplier, multiplier, 1           // shift right multiplier
    lsl multiplicand, multiplicand, 1           // shift left multiplicant
    add i, i, 1                     // i++
test:                               // Test for loop
    cmp i, 4                        // compare i < 4
    b.lt    forloop

resultPrint:                            // result print
    adrp    x0, printProduct                //1st arg
    add x0, x0, :lo12:printProduct          //1st arg
    mov w1, product                 //2nd
    mov w2, product                 //3rd
    bl  printf                      //call print

done:   mov w0, 0                       // Restore registers and returns to calling code
    ldp x29, x30, [sp], 16                  // Restore fp and lr from stack, post-incr sp
    ret                         // Return to caller:
```

exercise

- improve the program to fit negative numbers and numbers with more digits