Assignment 7

Wanzhang Sheng

March 28, 2014

MIPS version binary semaphores:

```
# Test
        .globl  main
main:
        li      $t0, 0
        sw      $t0, 0($sp)             # init lock as locked
        move    $a0, $sp
        jal     sem_post                # call sem_post
        lw      $t0, 0($sp)
        li      $a0, 1                  # error code 1
        bne     $t0, 1, done            # check if the lock is unlocked

        li      $t0, 1
        sw      $t0, 0($sp)             # init lock as unlocked
        move    $a0, $sp
        jal     sem_wait                # call sem_wait
        lw      $t0, 0($sp)
        li      $a0, 2                  # error code 2
        bne     $t0, 0, done            # check if the lock is locked

        li      $a0, 0                  # no error
        b       done                    # finished

done:
        li      $v0, 1
        syscall
        li      $v0, 10
        syscall


# Function:     sem_post
# Purpose:      To unlock a semaphore
```

```
#
# C Prototype:  int sem_post(sem_t *sem);
# Args:         &sem = a0
# Return val:   0 on success; -1 on error and the value is left unchanged.
#
        .globl  sem_post
sem_post:
        li      $t0, 1
        sw      $t0, 0($a0)     # set it to 1 for unlock
        li      $v0, 0          # return 0
        jr      $ra

# Function:     sem_wait
# Purpose:      To lock a semaphore
#
# C Prototype:  int sem_wait(sem_t *sem);
# Args:         &sem = a0
# Return val:   0 on success; -1 on error and the value is left unchanged.
#
        .globl  sem_wait
sem_wait:
        li      $t0, 0
        ll      $t1, 0($a0)             # linked load sem
        sc      $t0, 0($a0)             # change sem if not changed by others
        beq     $t0, $zero, sem_wait    # if t0 == 0 (changed), try again
        beq     $t1, 0, sem_wait        # if t1 == 0 (locked), try again

        li      $v0, 0                  # return 0
        jr      $ra
```

pth_msg_sem_mips.s

1

> Subtraction of two unsigned decimal integers:

```
    update = minu;
    for (digit = 0; digit < max_digits; digit++) {
       if (update[digit] < subt[digit]) {
          update[digit] += 10;
          i = digit + 1;
          while ((i < max_digits) && (update[i] == 0)) {
             update[i] = 9;
             i++;
          }
          if (i >= max_digits) exit(127); // Overflow
          update[i]--;
       }
```

```
      diff[digit] = update[digit] - subt[digit];
   }
```

If we use a marker to indicate the result is negative:

```
   update = minu;
   for (digit = 0; digit < max_digits; digit++) {
      if (update[digit] < subt[digit]) {
         update[digit] += 10;
         i = digit + 1;
         while ((i < max_digits) && (update[i] == 0)) {
            update[i] = 9;
            i++;
         }
         if (i >= max_digits) {
           // Fix the digits to negative number
           int j;
           for (j = 0; i < max_digits-1; j++)
             update[j] = 9 - update[j];
           // set the highest digit to -1 to indicate it is negative
           update[max_digits-1] = -1;
           return update;
         }
         update[i]--;
      }
      diff[digit] = update[digit] - subt[digit];
   }
```

2 3.1

> What is 5ED4 − 07A4 when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

0x5ED4 − 0x07A4 = 24276 − 1956 = 22320 = 5730

3 3.2

> What is 5ED4 − 07A4 when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

0x5ED4 − 0x07A4 = 24276 − 1956 = 22320 = 5730
Since 0x5 = 0101, first bit is 0, so these three numbers are all positive.

4 3.6

> “Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate $185-122$. Is there overflow, underflow, or neither?”

$185-122=63$
Since unsigned 8-bit integers range is 0 255, they are all in range.

5 3.7

> “Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate $185+122$. Is there overflow, underflow, or neither?”

Signed 8-bit integers range is -128 127, so 185 is actually -71.
$-71+122=51$
Which the result is in the range.

6 3.8

> “Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate $185-122$. Is there overflow, underflow, or neither?”

Since 185 is actually -71 in signed 8-bit integers.
$-71-122=-193$
Which the result is underflow.