CPSC 213, Winter 2015, Term 2 — Extra Questions **Solution**

Date: March 3, 2015; Instructor: Mike Feeley

1 (8 marks) **Loops and If.** The following assembly code computes `s = a[0]` where `a` is a global, static array of integers. Modify this code so that it computes the sum of all positive elements of the array where the size of the array is stored in a global int named `n`. Your solution should avoid unnecessary memory accesses where possible (e.g., inside of the loop). You may modify the code in place. Comment every line you add. Hint: notice that you have to add four things: (1) read the value of `n`, (2) turn part of this code into a loop, (3) exit the loop at the right time, and (4) only sum positive numbers; you might want to take these one at a time.

2 (6 marks) **Static Control Flow.** Give SM213 assembly code for the following C statements. Assume that `i` is a global variable of type `int`.

2a

```
if (i==0)
    i = 1;
else
    i = 2;
```

```
    ld $i, r0      # r0 = &i
    ld (r0), r1    # r1 = i
    beq r1, L0     # goto L0 if i==0
    ld $2, r2      # t_i = 2 if i !=0
    br L1          # goto L1
L0: ld $1, r2      # t_i = 1 if i==0
L1: st r2, (r0)    # i = t_i
```

2b

```
while (i!=0)
    i -= 1;
```

```
    ld $i, r0      # r0 = &i
    ld (r0), r1    # t_i = i
L0: beq r1, L2     # goto L1 if t_i == 0
    dec r1         # t_i--
    br L0          # goto L0
L1: st r1, (r0)    # i = t_i
```

3 (8 marks) **Dynamic Control Flow.** Give SM213 assembly code for the following C statements. Assume that `i` is a global variable of type `int`.

3a Using a jump table, the statement:

```
switch (i) {
    case 4:
        i = 0;
        break;
    case 6:
        i = 1;
        break;
    default:
        i = 2;
        break;
}
```

```
        ld $i, r0         # r0 = &i
        ld (r0), r1       # r1 = i
        ld $-4, r2        # r2 = -4
        add r2, r1        # r1 = i-4
        bgt r1, L0        # goto L0 if i > 4
        beq r1, L0        # goto L0 if i == 4
        br DEFAULT        # goto L0 if i < 4
L0:     ld $-2, r2        # r2 = -2
        add r1, r2        # r2 = i-6
        bgt r2, DEFAULT   # goto DEFAULT if i > 6
        ld $JT, r2        # r2 = JT
        j *(r2, r1, 4)    # goto jt[i-4]
CASE_4: ld $0, r2         # t_i = 0
        br L1             # goto L1
CASE_6: ld $1, r2         # t_i = 1
        br L1             # goto L1
DEFAULT: ld $2, r2        # t_i = 2
L1:     st r2, (r0)       # i = t_i

# The Jump Table
JT:     .long CASE_4
        .long DEFAULT
        .long CASE_6
```

3b Where the global variable `int (*bar)(void)` was previously declared, the statement:

```
bar();
```

```
ld $bar, r0  # r0 = &bar
gpc $2, r6   # r6 = return address
j *(r0)      # bar()
```

4 **(8 marks)** **Procedure Calls.** Give SM213 assembly for these statements. Assume the `i` is a global variable of type `int`, that `r5` stores the value of the stack pointer, and that arguments are passed on the stack.

4a

```
int foo (int i, int j) {
    return j;
}
```

```
ld 4(r6), r0  # r0 = j
j (r6)        # return j
```

4b

```
i = foo (1, 2);
```

```
deca r5       # make stack space for arg0
deca r5       # make stack space for arg1
ld $1, r0     # r0 = 1
st r0, 0(r5)  # arg0 = 1
ld $2, r0     # r0 = 2
st r0, 4(r5)  # arg1 = 2
gpc $6, r6    # r6 = return address
j foo         # t_i = foo (1,2)
inca r5       # free stack space for arg1
inca r5       # free stack space for arg0
ld $i, r1     # r1 = &i
st r0, (r1)   # r1 = t_i
```

5 **(12 marks)** Consider the following SM213 assembly code that implements a simple C procedure.

```
L0:  deca r5                 #  make stack space for saved ra
     st   r6, (r5)           #  store saved ra on stack
     ld   4(r5), r1          #  r1 = a
     ld   8(r5), r2          #  r2 = t_i = n
     ld   $0, r3             #  r3 = t_s = 0
L1:  bgt  r2, L2             #  goto L2 if t_i > 0
     br   L3                 #  goto L3 if t_i <= 0
L2:  dec  r2                 #  t_i --
     ld   (r1, r2, 4), r4    #  r4 = a[t_i]
     deca r5                 #  make stack space for arg
     st   r4, (r5)           #  arg = a[t_i]
     gpc  $2, r6             #  r6 = return address
     j    *16(r5)            #  t_j = f (a[t_i])
     inca r5                 #  free stack space for arg
     beq  r0, L1             #  goto L1 if t_j == 0
     add  r4, r3             #  t_s += a[i] if t_j != 0
     br   L1                 #  goto L1
L3:  mov  r3, r0             #  r0 = t_s
     ld   (r5), r6           #  r6 = saved return address
     inca r5                 #  free stack space for ra
     j    (r6)               #  return t_s
```

5a Comment every line in a way that illustrates the connection to corresponding C statements.

5b Give an equivalent C procedure (i.e., a procedure that may have compiled to this assembly code).

```
int foo (int* a, int n, int (*f)(int)) {
    int s = 0;
    for (int i=n-1; i>=0; i--)
        if f (a [i])
            s += a[i]
    return s;
}
```

```
        ld $a, r0              # r0 = &a = &[0]
        ld $0, r1              # r1 = temp_i = 0
        ld $0, r2              # r2 = temp_s = 0
        ld (r0, r1, 4), r3     # r3 = a[temp_i]
        add r3, r2             # temp_s = temp_s + a[temp_i]
        ld $s, r4              # r4 = &s
        st r2, (r4)            # s = temp_s
```

Added lines are numbered

```
            ld $a, r0             # r0 = &a = &[0]
            ld $0, r1             # r1 = temp_i = 0
            ld $0, r2             # r2 = temp_s = 0
[1]         ld $n, r5             # r5 = &n
[2]         ld (r5), r5           # r5 = n = temp_n
[3]     loop:
[4]         bgt r5, cont          # continue if temp_n > 0
[5]         br done               # exit look if temp_n <= 0
[6]     cont:
            ld (r0, r1, 4), r3    # r3 = a[temp_i]
[7]         dec r5                # temp_n --
[8]         inc r1                # temp_i ++
[9]         bgt r3, add           # goto add if a[temp_i] > 0
[10]        br  loop              # skip add & goto loop if a[temp_i] <= 0
[11]    add:
            add r3, r2            # temp_s += a[temp_i] if a[temp_i] < 0
[12]        br loop               # start next iteration of loop
[13]    done:
            ld $s, r4             # r4 = &s
            st r2, (r4)           # s = temp_s
```

6 **(10 marks)** **Writing Assembly Code**. Write SM213 assembly code that implements the following C program. Use labels for static addresses but do not include variable label declarations (i.e. "`.long`" lines). Show only the code for these two procedures. Do not implement a return from `callReplace()`; simply halt at the end of that procedure. Do not use the stack. Comment every line.

```
int* a;
int  searchFor, replaceWith, size, i;

void replace() {
    for (i=0; i<size; i++)
        if (a[i]==searchFor)
            a[i]=replaceWith;
}

void callReplace() {
    replace();
    // halt; do not return
}
```

```
replace:      ld   $size, r0          # r0 = &size
              ld   0x0(r0), r0        # r0 = size = i
              ld   $a, r1             # r1 = &a
              ld   0x0(r1), r1        # r1 = a
              ld   $searchFor, r2     # r2 = &searchFor
              ld   0x0(r2), r2        # r2 = searchFor
              not  r2                 # r2 = !searchFor
              inc  r2                 # r2 = -searchFor
              ld   $replaceWith, r3   # r3 = &replaceWith
              ld   0x0(r3), r3        # r3 = replaceWith
loop:         beq  r0, done           # goto done if i==0
              dec  r0                 # i--
              ld   (r1, r0, 4), r4    # r4 = a[i]
              add  r2, r4             # r4 = a[i] - searchFor
              beq  r4, match          # goto match   if a[i]==searchFor
              br   nomatch            # goto nomatch if a[i]!=searchFor
match:        st   r3, (r1, r0, 4)    # a[i] = replaceWith
nomatch:      br   loop               # goto loop
done:         j    0x0(r6)            # return
callReplace:  gpc  $0x6, r6           # ra = pc + 6
              j    replace            # replace()
              halt
```

7 **(10 marks)** Implement the following in SM213 assembly. Pass arguments on the stack. You can use a register for `c` instead of a local variable. **Comment every line.**

```
int countNotZero (int len, int* a) {
    int c=0;
    while (len>0) {
        len=len-1;
        if (a[len]!=0)
            c=c+1;
    }
    return c;
}
```

```
countZero: ld  0(r5), r1        # r1 = len
           ld  4(r5), r2        # r2 = a
           ld  $0, r0           # r0 = c
loop:      bgt r0, cont         # goto cont if len>0
           dec r1               # len = len - 1
           br  done             # goto done if len<=0
cont:      ld  (r2, r1, 4), r3  # r3 = a[len]
           beq r3, loop         # goto skip if a[len]==0
           inc r0               # c=c+1 if a[len]!=0
           br  loop             # goto loop
done:      j   (r6)             # return c
```