Computer Architecture (Sheet #1)

Marius Gavrilescu

1. R is an array of 9 char pointers (likely 9 strings).

P	4	20	x_P	$x_P + 4 \cdot i$
Q	2	4	x_Q	$x_Q + 2 \cdot i$
R	8	72	x_R	$x_R + 8 \cdot i$
S	8	80	x_S	$x_S + 8 \cdot i$
T	8	16	x_T	$x_T + 8 \cdot i$

2. ```
   char* binaryInt(int value) {
     int i;
     char *ret = malloc(33 * sizeof(char));
     for(i = 0 ; i < 32 ; i++)
       ret[i] = '0' + (value & (1 << (31 - i)) ? 1 : 0);
     ret[32] = 0;
     return ret;
   }
   ```

3. Let `z = *x` $\oplus$ `*y`. Note that `z` $\oplus$ `*x = *y` and `z` $\oplus$ `*y = *x`.

   First, we set `*y` to the value of `z`. Then we set `*x` to the value of `*x` $\oplus$ `z`, which is `*y`. Finally we set `*y` to the value of `*y` $\oplus$ `z`, which is `*x`. Note that in every "value of X" expression here we refer to the value of X before any assignments took place.

   This fails when `x = y`, as the first assignment will clobber the value of `*x`. The function can be fixed by changing `first <= last` to `first < last`, as this will avoid swapping equal pointers.

   We can rewrite the call as `swap(array + first, array + last)`.

4. ```
   %rbx            0x204
   0x204           0xBC
   $0x208          0x208
   (%rbx)          0xBC
   8(%rbx)         0x21
   7(%rax,%rcx)    0x24
   256(%rcx,%rdx)  0x21
   0x1FC(,%rcx,4)  0xBB
   (%rbx,%rcx,8)   0x21
   ```

5. ```
   addq %rcx,(%rax)          0x100 0x100
   subq %rcx,8(%rax)         0x108 0xA8
   imulq $16,(%rax,%rdx,8) 0x118 0x110
   incq 16(%rax)             0x110 0x14
   decq %rcx                 %rcx  0
   subq %rdx,%rax            %rax  0xFD
   ```
   ```

6.
```
leaq 9(%rdx),%rax        q+9
leaq (%rdx,%rbx),%rax    q+p
leaq (%rdx,%rbx,3),%rax  q+p*3
leaq 2(%rbx,%rbx,7),%rax p+p*7+2
leaq 0xE(,%rdx,3),%rax   q*3+14
leaq 6(%rbx,%rdx,7),%rax p+q*7+6
```

7. In order, %rdi, %rsi, %edx, %rcx, %r8w, %r9, 8(%rsp), 16(%rsp).
   They occupy 8, 8, 4, 8, 2, 8, 1, 8 bytes.

   Stack looks like (assume %rsp is 0x100):

```
0x110 $a4p
0x108 $a4
0x100 $ret
```

   where $ret is the return address (8 bytes), $a4 is the value of a4 (1 byte followed by 7 bytes of padding), and $a4p is the value of a4p (8 bytes).

8.
```
sumArrays: xorq %rax, %rax
loop:      decl %ecx
           cmpl $0, (%rdi, %ecx, 4)
           cmovge (%rsi, %ecx, 8), %r9
           cmovl  (%rdx, %ecx, 8), %r9
           add %r9, %rax
           cmpl $0, %ecx
           jnz loop
           ret
```