Kayla Cresswell
CS 6015
Popcount Assembly Analysis

FUNCTION: int popcount_1_data(uint64_t x)

```
_popcount_1_data:                      ## @popcount_1_data
        .cfi_startproc
## BB#0:
        pushq    %rbp
Lcfi0:
        .cfi_def_cfa_offset 16
Lcfi1:
        .cfi_offset %rbp, -16
        movq     %rsp, %rbp
Lcfi2:
        .cfi_def_cfa_register %rbp
        pushq    %r14
        pushq    %rbx
Lcfi3:
        .cfi_offset %rbx, -32
Lcfi4:
        .cfi_offset %r14, -24
        movl     $64, %ecx
        xorl     %eax, %eax
        movl     $257, %r8d         ## imm = 0x101
        movl     $258, %r9d         ## imm = 0x102
        movl     $259, %r10d         ## imm = 0x103
        movl     $260, %r11d         ## imm = 0x104
        movl     $261, %r14d         ## imm = 0x105
        movl     $262, %esi         ## imm = 0x106
        movl     $263, %ebx         ## imm = 0x107
        .p2align  4, 0x90
```

This code is setting up the stack frame with necessary elements.

mov quadward from rsp spot to rbp

current value of r14 and rbx are pushed onto the stack for later

set offset of rbx to -32
set offset of r14 to -24

movl means move if less, as it sets up all of these variables

imm are the constant values

Kayla Cresswell
CS 6015
Popcount Assembly Analysis

```
LBB0_1:                     ## =>This Inner Loop Header: Depth=1
            movl        %edi, %edx
            andl        $1, %edx
            addl        %edx, %eax
            bextrl      %r8d, %edi, %edx
            addl        %eax, %edx
            bextrl      %r9d, %edi, %eax
            addl        %edx, %eax
            bextrl      %r10d, %edi, %edx
            addl        %eax, %edx
            bextrl      %r11d, %edi, %eax
            addl        %edx, %eax
            bextrl      %r14d, %edi, %edx
            addl        %eax, %edx
            bextrl      %esi, %edi, %eax
            addl        %edx, %eax
            bextrl      %ebx, %edi, %edx
            addq        %rdx, %rax
            shrq        $8, %rdi
            addl        $-8, %ecx
            jne         LBB0_1
## BB#2:
                        ## kill: %EAX<def> %EAX<kill> %RAX<kill>
            popq        %rbx
            popq        %r14
            popq        %rbp
            retq
            .cfi_endproc
                        ## -- End function
            .section    __TEXT,__const
            .p2align    5           ## -- Begin function popcount_1_control
LCPI1_0:
            .quad       6               ## 0x6
            .quad       5               ## 0x5
            .quad       4               ## 0x4
            .quad       3               ## 0x3
LCPI1_2:
            .quad       10              ## 0xa
            .quad       9               ## 0x9
            .quad       8               ## 0x8
            .quad       7               ## 0x7
LCPI1_3:
            .quad       14              ## 0xe
            .quad       13              ## 0xd
            .quad       12              ## 0xc
            .quad       11              ## 0xb
LCPI1_4:
            .quad       18              ## 0x12
            .quad       17              ## 0x11
            .quad       16              ## 0x10
            .quad       15              ## 0xf
LCPI1_5:
            .quad       22              ## 0x16
            .quad       21              ## 0x15
            .quad       20              ## 0x14
            .quad       19              ## 0x13
LCPI1_6:
            .quad       26              ## 0x1a
            .quad       25              ## 0x19
            .quad       24              ## 0x18
            .quad       23              ## 0x17
LCPI1_7:
            .quad       30              ## 0x1e
            .quad       29              ## 0x1d
            .quad       28              ## 0x1c
            .quad       27              ## 0x1b
LCPI1_8:
 (THIS GOES ON UNTIL LCPI_15).....
```

move if less from edi to edx 5 – 8 byte registers

deallocate stack space

jump if not equal, keeps looping until it meets 64

kill EAX and RAX and pop everything off the stack

quad is an eight-byte value, setting values up to 64 while going through the loop

<u>Did the compiler do a good job?</u>

I think that the compiler is doing a good job in this function, because it is not pushing and popping things off of the stack unnecessarily. It cleans everything off of the stack at BB#2, which is right before the end of the function.

Kayla Cresswell
CS 6015
Popcount Assembly Analysis

FUNCTION: int popcount_4_data(uint64_t x)

```
_popcount_4_data:                  ## @popcount_4_data
            .cfi_startproc
## BB#0:
            pushq       %rbp
Lcfi8:
            .cfi_def_cfa_offset 16
Lcfi9:
            .cfi_offset %rbp, -16
            movq        %rsp, %rbp
Lcfi10:
            .cfi_def_cfa_register %rbp
            movl        %edi, %edx
            andl        $15, %edx
            leaq        _pop4(%rip), %rcx
            movq        %rdi, %rax
            shrq        $2, %rax
            andl        $60, %eax
            movl        (%rax,%rcx), %eax
            addl        (%rcx,%rdx,4), %eax
            movq        %rdi, %rdx
            shrq        $6, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $10, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $14, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $18, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $22, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $26, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $30, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $34, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $38, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $42, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $46, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $50, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            movq        %rdi, %rdx
            shrq        $54, %rdx
            andl        $60, %edx
            addl        (%rdx,%rcx), %eax
            shrq        $60, %rdi
            addl        (%rcx,%rdi,4), %eax
            popq        %rbp
            retq
            .cfi_endproc
                    ## -- End function
            .globl      _popcount_4_control     ## -- Begin function popcount_4_control
            .p2align    4, 0x90
```

push rbp onto the stack

mov results from rsp to rbp

move if less from edi to edx

load effective address of source into destination

shift right 4 bits

this repeats until loop terminates

Did the compiler do a good job?

The compiler did a better job pushing this data on than with the function 1 above, because it doesn't go in and out of the loop so many times, it just iterates until the end of the function. It seems like this would be more efficient with memory usage and with the stack because it doesn't have to load certain values each time the loop is entered.

Kayla Cresswell
CS 6015
Popcount Assembly Analysis

FUNCTION: int popcount_1_control(uint64_t x)

```
_popcount_1_control:                ## @popcount_1_control
        .cfi_startproc
## BB#0:
        pushq   %rbp
Lcfi5:
        .cfi_def_cfa_offset 16
Lcfi6:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Lcfi7:
        .cfi_def_cfa_register %rbp
        movl    %edi, %ecx
        andl    $1, %ecx
        movl    $257, %eax          ## imm = 0x101
        bextrl  %eax, %edi, %r8d
        movl    $258, %eax          ## imm = 0x102
        bextrl  %eax, %edi, %r9d
        vmovq   %rdi, %xmm0
        vpbroadcastq        %xmm0, %ymm3
        vpsrlvq LCPI1_0(%rip), %ymm3, %ymm0
        vpshufd $232, %ymm0, %ymm0      ## ymm0 = ymm0[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm0, %ymm0      ## ymm0 = ymm0[0,2,2,3]
        movl    LCPI1_1(%rip), %eax
        vmovd   %eax, %xmm1
        vpbroadcastd        %xmm1, %xmm2
        vpand   %xmm2, %xmm0, %xmm0
        vpsrlvq LCPI1_2(%rip), %ymm3, %ymm2
        vpsrlvq LCPI1_3(%rip), %ymm3, %ymm4
        vpshufd $232, %ymm4, %ymm4      ## ymm4 = ymm4[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm4, %ymm4      ## ymm4 = ymm4[0,2,2,3]
        vpshufd $232, %ymm2, %ymm2      ## ymm2 = ymm2[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm2, %ymm2      ## ymm2 = ymm2[0,2,2,3]
        vinserti128         $1, %xmm2, %ymm4, %ymm4
        vpbroadcastd        %xmm1, %ymm2
        vpand   %ymm2, %ymm4, %ymm1
        vpsrlvq LCPI1_4(%rip), %ymm3, %ymm4
        vpsrlvq LCPI1_5(%rip), %ymm3, %ymm5
        vpsrlvq LCPI1_6(%rip), %ymm3, %ymm6
        vpsrlvq LCPI1_7(%rip), %ymm3, %ymm7
        vpshufd $232, %ymm7, %ymm7      ## ymm7 = ymm7[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm7, %ymm7      ## ymm7 = ymm7[0,2,2,3]
        vpshufd $232, %ymm6, %ymm6      ## ymm6 = ymm6[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm6, %ymm6      ## ymm6 = ymm6[0,2,2,3]
        vinserti128         $1, %xmm6, %ymm7, %ymm6
        vpshufd $232, %ymm5, %ymm5      ## ymm5 = ymm5[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm5, %ymm5      ## ymm5 = ymm5[0,2,2,3]
        vpshufd $232, %ymm4, %ymm4      ## ymm4 = ymm4[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm4, %ymm4      ## ymm4 = ymm4[0,2,2,3]
        vinserti128         $1, %xmm4, %ymm5, %ymm4
        vpand   %ymm2, %ymm4, %ymm4
        vpand   %ymm2, %ymm6, %ymm5
        vpsrlvq LCPI1_8(%rip), %ymm3, %ymm6
        vpsrlvq LCPI1_9(%rip), %ymm3, %ymm7
        vpsrlvq LCPI1_10(%rip), %ymm3, %ymm8
        vpsrlvq LCPI1_11(%rip), %ymm3, %ymm9
        vpsrlvq LCPI1_14(%rip), %ymm3, %ymm10
        vpsrlvq LCPI1_15(%rip), %ymm3, %ymm11
        vpshufd $232, %ymm11, %ymm11    ## ymm11 =
ymm11[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm11, %ymm11    ## ymm11 = ymm11[0,2,2,3]
        vpshufd $232, %ymm10, %ymm10    ## ymm10 =
ymm10[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm10, %ymm10    ## ymm10 = ymm10[0,2,2,3]
        vinserti128         $1, %xmm10, %ymm11, %ymm10
        vpsrlvq LCPI1_12(%rip), %ymm3, %ymm11
        vpsrlvq LCPI1_13(%rip), %ymm3, %ymm3
        vpshufd $232, %ymm3, %ymm3      ## ymm3 = ymm3[0,2,2,3,4,6,6,7]
        vpermq  $232, %ymm3, %ymm3      ## ymm3 = ymm3[0,2,2,3]
        vpshufd $232, %ymm11, %ymm11    ## ymm11 =
```

push rbp on the stack

move edi if it is less

set immediate constant

mov quadward rdi to xmm0

load integer and broadcast

shuffle packed bytes (x & 0xf code)

bit shift right 4

....repeat till loop terminates....

<u>Did the compiler do a good job?</u>

I think that the compiler did a good job with this function as well. It is setting all the values, then permutating and shuffling, then inserting and finally broadcasting. It does a better job than setting the data in 1 because the LCPI1_... through 15 is doing the bit shifting all in line right after each other. This seems very efficient to memory and with data accesses from RAM or from disk.