ECS 40 **Program #4 (50 pts, 2.5 hours)** **Winter 2016**

Due: Wednesday, February 11th, 11:59 PM. Executable Name: CPU.out

Filenames (case sensitive): authors.csv decoder.cpp, decoder.h, instruction.cpp, instruction.h, labels.cpp, labels.h, main.cpp, reader.cpp, reader.h, registers.cpp, registers.h, and Makefile.

For this assignment you will be adding more functionality to Program #3. Program #4 will process assembly code that involves arrays, function calls, and branching. As usual, all implementation code should be in the .cpp files. You are free (and encouraged) to use my source code for p3 as your starting point. In any case, you may wish to refer to my code if you do not understand some of the references in the instructions. There are two new assembly files, test4.s and test5.s. They and their C files are attached.

Both files rely on using labels to indicate addresses in the code, e.g. `jg .L2`, and `call _Z3fooii`. Your program will need to be able to find a label and determine its address. You will have a Labels class object that will hold the labels with their addresses.

There are two new registers, edx, and flags. edx is a data register often used to hold an index. For arrays, we have a new addressing mode, scaled index mode. There is only one example this week: -72(%ebp, %edx, 4). The address of the value sought would be the value in ebp + four times the value in edx minus 72, i.e., %ebp + 4 * %edx – 72. The four is used because it is the size of an int.

The sixth register is the flags register. For this assignment, we will only use two bits of the flags int. The sign flag, SF, is bit 7 (counting with bit zero as the least significant bit) and is set to one when a result is negative, else it is zero. The zero flag, ZF, is bit 6, and is set to one when a result is zero, else it is set to zero. There are no instructions that explicitly set or read the flags register, but many instructions implicitly use it. **cmpl, addl, incl, andl**, and **subl** all can change the flags. The two conditional jump instructions, **jg,** and **jle** rely on the values of the flags to determine whether the eip should be changed.

You should alter your code to process test4.s, and then add the code to make test5.s work. Your code should compile without warnings, and run properly with test2.s after each step.

1. test4.s
   1.1. Change the Reader::read() method to an overloaded extraction (>>) operator, and have main() call it by having main() opening the file.
   1.2. Create a Labels class, and create a labels object in main().
      1.2.1. class Labels has an array of 100 Instructions, and an int count.
      1.2.2. Intel assembly language labels start their lines with either ".L",or an underscore and end with a colon. When storing a label you should strip the colon from it.
      1.2.3. main() should call an overloaded extraction operator (>>) to read the labels into its array from the file. This means the file will be read twice, once by Labels' operator>>, and once by the Reader's operator>>. To rewind an open file you will need to use seekg(), but before that you will need to call clear() to the failbit of the file. The Reader's operator>> code is a good place to use as a starting point.
      1.2.4. Add a find() method that takes a string and returns its address.
   1.3. Alter the Registers class in the following manner.
      1.3.1. Add the edx and flags registers to regs, and add edx and flags to the enum.
      1.3.2. Change Registers::initialize() to a default constructor, and just initialize memory[1000] in main().
         1.3.2.1. Initialize edx to zero, and flags to 0xC0.
      1.3.3. Replace the print() method with an overloaded insertion operator (<<) that also prints the edx and flags registers. main() will now print out the instruction's info.
      1.3.4. Move the enum into the public part of the Registers class. This would allow other classes to safely have their own ebx, esp … constants.
         1.3.4.1. Change all parameters that were RegName to Registers::RegName
         1.3.4.2. Change all code outside the Registers class that references the enums to Registers::*enum*, e.g. Registers::eax.
      1.3.5. Add a += operator that takes an int as its parameter, and adds it to the existing value of esp. It returns the new value of esp. Note that this can be used to add or subtract values from the esp. Use of this operator should replace the Registers::set(…Registers::get()) esp calls in decoder.cpp
      1.3.6. address() will have to altered to handle labels and scaled index addressing mode. It will now have a Labels object has an additional const reference parameter. At this point, scaled index addressing mode can be differentiated because it is the only addressing mode that involves two commas. The actual registers

involved in scaled index addressing mode cannot be assumed to be always the same. Since functions not using switch() should have bodies of less than 35 lines, the address() method will become too large. You should create additional new functions to handle the complexity of addressing modes. I spun off a private helper function called stringToRegNum(). Beware that since parse() uses strtok(), address() and its subroutines cannot.

1.3.7. Alter the Decoder::parse() by passing it a Labels object that is passed to address().

1.4. Alter the existing Decoder **addl, subl, andl** functions to change the bits in the flags register based on their results.

1.4.1. Add a setFlags() method to Registers that will set the flags based on the int parameter. Note that the parameter is the result of the operation, not the value to which the flags should be set.

1.5. Decoder now has these additional instructions to implement, and call from Decoder::execute()

1.5.1. Add appropriate methods to the Registers class to read a specific flag as a bool.

1.5.2. **cmpl** *operand1*, *operand2* : sets the flags register based on subtracting *operand1* from *operand2*. The result is not stored anywhere.

1.5.3. **incl** *operand* : increments *operand*, and sets the SF and ZF accordingly.

1.5.4. **jg** *address :* jump if greater, so place *address* in eip if SF is clear and ZF is clear.

1.5.5. **jle** *address :* jump if less than or equal, so place *address* in eip if SF is set or ZF is set.

1.5.6. **jmp** *address :* place *address* in the eip.

1.5.7. **leal** *operand1, operand2* : load the effective address of *operand1* into *operand2*.

1.5.7.1. For example, the effective address of –16(%ebp) is the value held in ebp minus 16. The address() method will not help you with determining the effective address of *operand1*.

1.5.7.2. You will have to write your own parsing code for *operand1*. However, for this assignment, you may assume that it will always use the value held in ebp. So you need only parse out the number in front of the register.

1.6. Your program should now be able to run test4.s

2. test5.s

2.1. The Reader operator>> will have to ignore labels that start with an underscore.

2.2. The Labels operator>> will now have to detect “main:”. main(), should set the eip to the address of “main:”.

2.3. Decoder will now have these additional instruction to implement, can call from Decoder::execute()

2.3.1. **call** *address* : pushes the value in eip on the stack and puts *address* in eip. To push on the stack you decrement esp by four, and then copy eip to the location that esp now indicates. Make sure your **pushl** also does this in the same order, first decrement, then copy.

2.3.2. **sall** *operand1, operand2* : shift *operand2* left *operand1* bits, and changes the flags accordingly.

Further specifications:

1.1. const must be used wherever possible in function headings. This includes parameters, return types, and functions themselves. Note that there is no need to label parameters and return types that are passed by value as const.

1.2. You may assume that all input will be valid, and not require any form of range checking.

1.3. Your Makefile file must contain six pairs of compiling lines, and one pair of linking lines. You must use g++ with the –g –Wall –ansi options for compiling and linking. You will lose one point for each warning.

1.4. You will find test?.c, test?.S, and my own executable in ~ssdavis/40/p4 Thursday morning. You will find my p3 source code available in ~ssdavis/40/p3/SeansSrc on Thursday morning.

```
[ssdavis@lect1 p4]$ cat test4.c
#include <stdio.h>

int main()
{
  int a, b, c[10], i;
  a = 197;

  if(a < 100)
    b = 134;
  else
    b = 26;

  for(i = 0; i < 3; i++)
    c[i] = a + b;

  a = c[5] + 18;
  return b;
}
[ssdavis@lect1 p4]$
```

```
[ssdavis@lect1 p4]$ cat test4.s
        .file   "test4.c"
        .text
        .align 2
.globl main
        .type   main,@function
main:
.LFB2:
        pushl   %ebp
.LCFI0:
        movl    %esp, %ebp
.LCFI1:
        subl    $88, %esp
.LCFI2:
        andl    $-16, %esp
        movl    $0, %eax
        subl    %eax, %esp
        movl    $197, -12(%ebp) # a = 197
        cmpl    $99, -12(%ebp)  # compare a and 99
        jg      .L2     # if a greater then goto .L2
        movl    $134, -16(%ebp) # b = 134
        jmp     .L3     # goto .L3
.L2:
        movl    $26, -16(%ebp)  # b = 26
.L3:        movl    $0, -76(%ebp)  # i = 0
.L4:
        cmpl    $2, -76(%ebp)# compare i and 2
        jle     .L7 # if i <= 2 goto .L7
        jmp     .L5 # goto .L5
.L7:
        movl    -76(%ebp), %edx  # edx = i
        movl    -16(%ebp), %eax  # eax = b
        addl    -12(%ebp), %eax  # eax += a
        movl    %eax, -72(%ebp,%edx,4)
        leal    -76(%ebp), %eax     # eax = &i
        incl    (%eax)              # i++
        jmp     .L4
.L5:
        movl    -52(%ebp), %eax  # eax = c[5]
        addl    $18, %eax        # eax += 18
        movl    %eax, -12(%ebp)  # a = eax
        movl    -16(%ebp), %eax  # return b
        leave
        ret
.LFE2:
.Lfe1:
        .size   main,.Lfe1-main
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.2.3 20030502 (Red Hat
Linux 3.2.3-49)"

[ssdavis@lect1 p4]$
```

```
[ssdavis@lect1 p4]$ cat test5.c
#include <stdio.h>

int foo(int x, int y)
{
  return x + y;
} // foo

void twice(int *z)
{
  *z = *z * 2;
}

int main()
{
  int a, b, c;
  a = 7;
  b = 15;
  c = foo(a , 31);
  twice(&b);

  return c;
}
[ssdavis@lect1 p4]$


[ssdavis@lect1 p4]$ cat test5.s
        .file   "test5.c"
        .text
        .align 2
.globl _Z3fooii
        .type   _Z3fooii,@function
_Z3fooii:
.LFB2:
        pushl   %ebp
.LCFI0:
        movl    %esp, %ebp
.LCFI1:
        movl    12(%ebp), %eax
        addl    8(%ebp), %eax
        leave
        ret
.LFE2:
.Lfe1:
        .size   _Z3fooii,.Lfe1-_Z3fooii
        .align 2
.globl _Z5twicePi
        .type   _Z5twicePi,@function
_Z5twicePi:
.LFB4:
        pushl   %ebp
.LCFI2:
        movl    %esp, %ebp
.LCFI3:
        movl    8(%ebp), %edx
        movl    8(%ebp), %eax
  movl    (%eax), %eax
        sall    $1, %eax
        movl    %eax, (%edx)
        leave
        ret
.LFE4:
.Lfe2:
        .size   _Z5twicePi,.Lfe2-_Z5twicePi
        .align 2
.globl main
        .type   main,@function
main:
.LFB6:
        pushl   %ebp
.LCFI4:
        movl    %esp, %ebp
.LCFI5:
        subl    $24, %esp
.LCFI6:
        andl    $-16, %esp
        movl    $0, %eax
        subl    %eax, %esp
        movl    $7, -4(%ebp)
        movl    $15, -8(%ebp)
        subl    $8, %esp
        pushl   $31
        pushl   -4(%ebp)
.LCFI7:
        call    _Z3fooii
        addl    $16, %esp
        movl    %eax, -12(%ebp)
        subl    $12, %esp
        leal    -8(%ebp), %eax
        pushl   %eax
        call    _Z5twicePi
        addl    $16, %esp
        movl    -12(%ebp), %eax
        leave
        ret
.LFE6:
.Lfe3:
        .size   main,.Lfe3-main
        .section        .note.GNU-stack,"",@progbits
        .ident  "GCC: (GNU) 3.2.3 20030502 (Red Hat
Linux 3.2.3-49)"
[ssdavis@lect1 p4]$
```

```
[ssdavis@lect1 p4]$ CPU.out test4.s
pushl %ebp           eip: 104 eax:   0 ebp: 996 esp: 996 edx:   0 flags: 192
movl %esp, %ebp      eip: 108 eax:   0 ebp: 996 esp: 996 edx:   0 flags: 192
subl $88, %esp       eip: 112 eax:   0 ebp: 996 esp: 908 edx:   0 flags:   0
andl $-16, %esp      eip: 116 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
movl $0, %eax        eip: 120 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
subl %eax, %esp      eip: 124 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
movl $197, -12(%ebp) eip: 128 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
cmpl $99, -12(%ebp)  eip: 132 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
jg .L2               eip: 144 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
movl $26, -16(%ebp)  eip: 148 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
movl $0, -76(%ebp)   eip: 152 eax:   0 ebp: 996 esp: 896 edx:   0 flags:   0
cmpl $2, -76(%ebp)   eip: 156 eax:   0 ebp: 996 esp: 896 edx:   0 flags: 128
jle .L7              eip: 164 eax:   0 ebp: 996 esp: 896 edx:   0 flags: 128
movl -76(%ebp), %edx eip: 168 eax:   0 ebp: 996 esp: 896 edx:   0 flags: 128
movl -16(%ebp), %eax eip: 172 eax:  26 ebp: 996 esp: 896 edx:   0 flags: 128
addl -12(%ebp), %eax eip: 176 eax: 223 ebp: 996 esp: 896 edx:   0 flags:   0
movl %eax, -72(%ebp,%edx,4) eip: 180 eax: 223 ebp: 996 esp: 896 edx:   0 flags:   0
leal -76(%ebp), %eax eip: 184 eax: 920 ebp: 996 esp: 896 edx:   0 flags:   0
incl (%eax)          eip: 188 eax: 920 ebp: 996 esp: 896 edx:   0 flags:   0
jmp .L4              eip: 152 eax: 920 ebp: 996 esp: 896 edx:   0 flags:   0
cmpl $2, -76(%ebp)   eip: 156 eax: 920 ebp: 996 esp: 896 edx:   0 flags: 128
jle .L7              eip: 164 eax: 920 ebp: 996 esp: 896 edx:   0 flags: 128
movl -76(%ebp), %edx eip: 168 eax: 920 ebp: 996 esp: 896 edx:   1 flags: 128
movl -16(%ebp), %eax eip: 172 eax:  26 ebp: 996 esp: 896 edx:   1 flags: 128
addl -12(%ebp), %eax eip: 176 eax: 223 ebp: 996 esp: 896 edx:   1 flags:   0
movl %eax, -72(%ebp,%edx,4) eip: 180 eax: 223 ebp: 996 esp: 896 edx:   1 flags:   0
leal -76(%ebp), %eax eip: 184 eax: 920 ebp: 996 esp: 896 edx:   1 flags:   0
incl (%eax)          eip: 188 eax: 920 ebp: 996 esp: 896 edx:   1 flags:   0
jmp .L4              eip: 152 eax: 920 ebp: 996 esp: 896 edx:   1 flags:   0
cmpl $2, -76(%ebp)   eip: 156 eax: 920 ebp: 996 esp: 896 edx:   1 flags:  64
jle .L7              eip: 164 eax: 920 ebp: 996 esp: 896 edx:   1 flags:  64
movl -76(%ebp), %edx eip: 168 eax: 920 ebp: 996 esp: 896 edx:   2 flags:  64
movl -16(%ebp), %eax eip: 172 eax:  26 ebp: 996 esp: 896 edx:   2 flags:  64
addl -12(%ebp), %eax eip: 176 eax: 223 ebp: 996 esp: 896 edx:   2 flags:   0
movl %eax, -72(%ebp,%edx,4) eip: 180 eax: 223 ebp: 996 esp: 896 edx:   2 flags:   0
leal -76(%ebp), %eax eip: 184 eax: 920 ebp: 996 esp: 896 edx:   2 flags:   0
incl (%eax)          eip: 188 eax: 920 ebp: 996 esp: 896 edx:   2 flags:   0
jmp .L4              eip: 152 eax: 920 ebp: 996 esp: 896 edx:   2 flags:   0
cmpl $2, -76(%ebp)   eip: 156 eax: 920 ebp: 996 esp: 896 edx:   2 flags:   0
jle .L7              eip: 160 eax: 920 ebp: 996 esp: 896 edx:   2 flags:   0
jmp .L5              eip: 192 eax: 920 ebp: 996 esp: 896 edx:   2 flags:   0
movl -52(%ebp), %eax eip: 196 eax:   0 ebp: 996 esp: 896 edx:   2 flags:   0
addl $18, %eax       eip: 200 eax:  18 ebp: 996 esp: 896 edx:   2 flags:   0
movl %eax, -12(%ebp) eip: 204 eax:  18 ebp: 996 esp: 896 edx:   2 flags:   0
movl -16(%ebp), %eax eip: 208 eax:  26 ebp: 996 esp: 896 edx:   2 flags:   0
leave                eip: 212 eax:  26 ebp: 996 esp: 1000 edx:   2 flags:   0
ret                  eip:   0 eax:  26 ebp: 996 esp: 1004 edx:   2 flags:   0
[ssdavis@lect1 p4]$
```

```
[ssdavis@lect1 p4]$ CPU.out test5.s
pushl %ebp           eip: 164 eax:   0 ebp: 996 esp: 996 edx:   0 flags: 192
movl %esp, %ebp      eip: 168 eax:   0 ebp: 996 esp: 996 edx:   0 flags: 192
subl $24, %esp       eip: 172 eax:   0 ebp: 996 esp: 972 edx:   0 flags:   0
andl $-16, %esp      eip: 176 eax:   0 ebp: 996 esp: 960 edx:   0 flags:   0
movl $0, %eax        eip: 180 eax:   0 ebp: 996 esp: 960 edx:   0 flags:   0
subl %eax, %esp      eip: 184 eax:   0 ebp: 996 esp: 960 edx:   0 flags:   0
movl $7, -4(%ebp)    eip: 188 eax:   0 ebp: 996 esp: 960 edx:   0 flags:   0
movl $15, -8(%ebp)   eip: 192 eax:   0 ebp: 996 esp: 960 edx:   0 flags:   0
subl $8, %esp        eip: 196 eax:   0 ebp: 996 esp: 952 edx:   0 flags:   0
pushl $31            eip: 200 eax:   0 ebp: 996 esp: 948 edx:   0 flags:   0
pushl -4(%ebp)       eip: 204 eax:   0 ebp: 996 esp: 944 edx:   0 flags:   0
call _Z3fooii        eip: 100 eax:   0 ebp: 996 esp: 940 edx:   0 flags:   0
pushl %ebp           eip: 104 eax:   0 ebp: 996 esp: 936 edx:   0 flags:   0
movl %esp, %ebp      eip: 108 eax:   0 ebp: 936 esp: 936 edx:   0 flags:   0
movl 12(%ebp), %eax  eip: 112 eax:  31 ebp: 936 esp: 936 edx:   0 flags:   0
addl 8(%ebp), %eax   eip: 116 eax:  38 ebp: 936 esp: 936 edx:   0 flags:   0
leave                eip: 120 eax:  38 ebp: 996 esp: 940 edx:   0 flags:   0
ret                  eip: 208 eax:  38 ebp: 996 esp: 944 edx:   0 flags:   0
addl $16, %esp       eip: 212 eax:  38 ebp: 996 esp: 960 edx:   0 flags:   0
movl %eax, -12(%ebp) eip: 216 eax:  38 ebp: 996 esp: 960 edx:   0 flags:   0
subl $12, %esp       eip: 220 eax:  38 ebp: 996 esp: 948 edx:   0 flags:   0
leal -8(%ebp), %eax  eip: 224 eax: 988 ebp: 996 esp: 948 edx:   0 flags:   0
pushl %eax           eip: 228 eax: 988 ebp: 996 esp: 944 edx:   0 flags:   0
call _Z5twicePi      eip: 124 eax: 988 ebp: 996 esp: 940 edx:   0 flags:   0
pushl %ebp           eip: 128 eax: 988 ebp: 996 esp: 936 edx:   0 flags:   0
movl %esp, %ebp      eip: 132 eax: 988 ebp: 936 esp: 936 edx:   0 flags:   0
movl 8(%ebp), %edx   eip: 136 eax: 988 ebp: 936 esp: 936 edx: 988 flags:   0
movl 8(%ebp), %eax   eip: 140 eax: 988 ebp: 936 esp: 936 edx: 988 flags:   0
movl (%eax), %eax    eip: 144 eax:  15 ebp: 936 esp: 936 edx: 988 flags:   0
sall $1, %eax        eip: 148 eax:  30 ebp: 936 esp: 936 edx: 988 flags:   0
movl %eax, (%edx)    eip: 152 eax:  30 ebp: 936 esp: 936 edx: 988 flags:   0
leave                eip: 156 eax:  30 ebp: 996 esp: 940 edx: 988 flags:   0
ret                  eip: 232 eax:  30 ebp: 996 esp: 944 edx: 988 flags:   0
addl $16, %esp       eip: 236 eax:  30 ebp: 996 esp: 960 edx: 988 flags:   0
movl -12(%ebp), %eax eip: 240 eax:  38 ebp: 996 esp: 960 edx: 988 flags:   0
leave                eip: 244 eax:  38 ebp: 996 esp: 1000 edx: 988 flags:   0
ret                  eip:   0 eax:  38 ebp: 996 esp: 1004 edx: 988 flags:   0
[ssdavis@lect1 p4]$
```