```
#include <stdio.h>

int main() {
  int x = 5;
  double y = 10.5;
  char z = 'a';
  double result;
  result = y+x;
  printf("%5.2f\n", result);
  return 0;
}
```

I had to start by trying to figure out what each instruction did, then I broke the program into segments for assembly, and finally attempted to mark the assembly code correspondence with the original C program.

Assembly File (.s) from running gcc -s FILENAME on Unix server:

```
        .file   "HW5.c"             **Name of c file assembly language was generated for**
        .section .rodata
.LC1:                               **Constants**
        .string "%5.2f\n"           **String constant**
        .align 8
.LC0:                               **Constant used to hold the floating point number 10.5**
        .long   0
        .long   1076166656
        .text
.globl main
        .type   main, @function
main:                               **ASSEMBLY INSTRUCTIONS***
        leal    4(%esp), %ecx       **Saves the value lying 4 bits in the stack (esp register) in the ecx register**
        andl    $-16, %esp          **Align the stack for 16bit addresses?***
        pushl   -4(%ecx)            **Push the value stored in ecx previously back onto the stack after stack has been aligned**
        pushl   %ebp                **Push contents of base pointer register onto stack**
        movl    %esp, %ebp          **Move contents of stack pointer register to base pointer register (for use in operation)***
        pushl   %ecx                **Push contents of ecx (gen purpose register) onto stack**
        subl    $52, %esp           **Reserve 52 bits on stack for following operations**
        movl    $5, -36(%ebp)       **Move decimal 5 onto the base pointer register (for use in operation) offset by 36 bytes**
        fldl    .LC0                **Load long (constant) for use as floating point number**
        fstpl   -32(%ebp)
        movb    $97, -17(%ebp)      **Move decimal 97 (ASCII 'a') onto base pointer register**
        fildl   -36(%ebp)
        faddl   -32(%ebp)
        fstpl   -16(%ebp)
        fldl    -16(%ebp)
        fstpl   4(%esp)
        movl    $.LC1, (%esp)       **Move string (constant from .LC1) onto stack for printing***
        call    printf              **print the string**
        movl    $0, %eax
        addl    $52, %esp
        popl    %ecx                **Pop item off of stack and store in ecx register**
        popl    %ebp                **Pop item off of stack and store in base pointer register**
        leal    -4(%ecx), %esp
        ret                         **Return**
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
        .section .note.GNU-stack,"",@progbits
```

Research:

This appears to be the assembly code for an x86 architecture.

eax = general purpose register

ecx = general purpose register

esp = stack pointer (pointer to the stack), relative to SS (segment register)
ebp = base pointer register. Appears to usually be use to complete operations before the final result is pushed onto the stack register (esp).

Instructions:
leal = Load Effective Address
pushl = push operand (contents of register) onto stack
pool = pop element from stack and store in specified location.
andl = aligns the stack.
subl = used to reserve space on the stack, stack grows downward.
mlvl = move contents from first operand, to second operand
ret = return, usually sets return address to appropriate register

Following is the assembly code for different segments of the original program. I created small segments and generated the assembly code for them to find which assembly code segments were responsible for different c program statements.

————————————————————————————————
Assembly code for char z = 'a':

```
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $16, %esp
        movb    $97, -5(%ebp)
        addl    $16, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
        .section .note.GNU-stack,"",@progbits
```

————————————————————————————————
Assembly code for int x = 5:

```
        .file   "Test.c"
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $16, %esp
        movl    $5, -8(%ebp)
        addl    $16, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
        .section .note.GNU-stack,"",@progbits
```

————————————————————————————————
Assembly code for double y = 10.5:

```
        .file   "Test.c"
        .section .rodata
        .align 8
.LC0:                                                   **.LC0 is used for floating point**
        .long   0
        .long   1076166656
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $20, %esp
        fldl    .LC0
        fstpl   -16(%ebp)
        addl    $20, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
        .section .note.GNU-stack,"",@progbits
```

———————————————————————————————

Assembly code for adding x and y, declaring result, storing sum in result:

```
        .file   "Test.c"
        .section .rodata
        .align 8
.LC0:
        .long   0
        .long   1076166656
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $36, %esp
        movl    $5, -28(%ebp)
        fldl    .LC0
        fstpl   -24(%ebp)
        fildl   -28(%ebp)
        faddl   -24(%ebp)
        fstpl   -16(%ebp)
        addl    $36, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
        .section .note.GNU-stack,"",@progbits
```

———————————————————————————————

Assembly for using printf(“%5.2f”, y) to print y = 10.5:

```
        .file   "Test.c"
        .section .rodata
```

```
.LC1:                           **.LC1 is used for String constant**
        .string "%5.2f"
        .align 8
.LC0:
        .long   0
        .long   1076166656
        .text
.globl main
        .type   main, @function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $36, %esp
        fldl    .LC0
        fstpl   -16(%ebp)
        fldl    -16(%ebp)
        fstpl   4(%esp)
        movl    $.LC1, (%esp)
        call    printf
        addl    $36, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
        .section .note.GNU-stack,"",@progbits
```

Full Assembly File divided into segments (corresponding to C program):

```
        .file   "HW5.c"
        .section .rodata
.LC1:                               **STRING CONSTANT**
        .string "%5.2f\n"
        .align 8
.LC0:                               **FLOAT CONSTANT**
        .long   0
        .long   1076166656
        .text
.globl main
        .type   main, @function
main:                               **ASSEMBLY INSTRUCTIONS**
        leal    4(%esp), %ecx       **STACK PREPARATION**
        andl    $-16, %esp          —align stack
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx                **END STACK PREP**
        subl    $52, %esp           **RESERVE SPACE ON STACK**
        movl    $5, -36(%ebp)       ** X = 5 **
        fldl    .LC0                **FLOATING POINT ( Y = 10.5 )**
        fstpl   -32(%ebp)
        movb    $97, -17(%ebp)      **ASCII 97 = 'a', ( Z = 'a' )**
        fildl   -36(%ebp)
        faddl   -32(%ebp)           **ADD X AND Y**
        fstpl   -16(%ebp)
        fldl    -16(%ebp)
        fstpl   4(%esp)
        movl    $.LC1, (%esp)       **MOVE STRING CONSTANT ONTO STACK ("%5.2f")**
        call    printf              **PRINT THE STRING**
```

```
movl    $0, %eax
addl    $52, %esp           **I think this moves the stack to the original position after function is complete (subl reserved 52 byte space)**
popl    %ecx                **FINISHING PROGRAM (RESET STACK)**
popl    %ebp
leal    -4(%ecx), %esp
ret                         **RETURN, END PROGRAM**
.size   main, .-main
.ident  "GCC: (GNU) 4.1.2 20080704 (Red Hat 4.1.2-54)"
.section .note.GNU-stack,"",@progbits
```