1. Consider the cache example on pp.19-20 of the notes on “computer engines.”

(a) (10) When a cache ______ occurs, the CPU will need to read a ______ from memory, which consists of ______ (fill in a number) bytes.

(b) (10) Consider the scenario described in footnote 22, for Version I (not II) of the code, at the time immediately after we have added X[0][31] to Sum. (Suppose the latter is stored in a register, thus not affecting the cache.) How many misses—if any—have occurred up to this point? Explain fully in order to receive full credit.

2. (25) Suppose in Homework IV we also have a function totfr(). It has no arguments, but it returns an int value which is the total number of bytes currently free in all of heap. Fill in the blanks in the following code for this function:

```c
...totfr:
    ___________________ # fill in one instruction
    movl $0, %eax
    movl _______, %ebx
    top:
    cmp __________, (%ebx)
    jz done
    addl ______, %ebx
    addl __________, __________
    addl $4, %ebx
    jmp top
    done:
    ___________________ # fill in one instruction
    ret
```

3. This problem concerns the example in Sec. 3 of the unit on machine language.

(a) (5) Suppose on line 26 the instruction were to use EDX instead of EBX. What would the machine code 0xbb000000 change to in the output of `as -a`?

(b) (10) Suppose line 34 were to have a label verydone, and the instruction in line 32 were jnz verydone. What would the machine code 0x75f8 for that instruction change to in the output of `as -a`?

In the remaining parts, assume that ld arranges for the .data and .text segments to be loaded beginning at 0x200 and 0x8000, respectively. Do not assume any changes mentioned in parts (a) and (b) above.

(c) (10) Consider the DEC instruction in line 31. List (in hex) all values which will pass through the MAR during this instruction (including all of Steps A, B and C, though do not break down your answer by step), and then do the same for the MDR. Ignore effects of prefetching and caches.

(d) (5) Would the machine language for any of the instructions change from what is listed in the output of `as -a`? For example, in line 30, would the 0x83c104 change to, say 0x83c304? Either give an example of an instruction in this program whose machine code would change, and state what the new machine code would be, or explain clearly why none of the machine language in the output of `as -a` would change.

4. This problem concerns the function sum() on p.10 of the notes unit on subroutines.

(a) (5) The compiler wasn’t very efficient here. Give two instructions it could have used in place of subl $8, %esp movl $0, -8(%ebp) movl $0, -4(%ebp)

(b) (10) Give specific assembly code, consisting of just one instruction, that the compiler will likely produce from the line return s

5. (10) Suppose in a C program main() calls f(), which in turn calls g(). Suppose also that the declaration of f() begins with

```c
int f(int x, int y) {
  int u,v,w;
  ...
}
```

Also, g() has no arguments and has only one local variable, z, of type int.

Write a single C statement, which would be in g(), which would set f()’s local variable w to 12. Your statement should involve &z

Solutions:

1.a. miss; block; 64

1.b. The key point is that when the read of X[0][0] causes the first cache miss, only the array through X[0][30] will be brought in. The read of X[0][31] will then cause a second miss. The answer is 2.

2.
# make totfr accessible from the C code
.globl totfr
totfr:
    # save old EBX, since we will write to it
    pushl %ebx
    movl $0, %eax
    # map contains the locations and sizes
    # of the free chunks, so point
    # EBX to it
    movl $map, %ebx
top:
    # map uses -1 as a termination signal
    cmpl $-1, (%ebx)
jz done
    # no termination, so move to the next
    # element of map, which gives the
    # size of this free chunk
    addl $4, %ebx
    # read that size
    addl (%ebx), %eax
    # go to the next element of map, which
    # we will check for termination
    addl $4, %ebx
    jmp top
done:
    # restore old EBX
    popl %ebx
    ret

3.a. 0xba00000000
3.b. 0x7506
3.c. The instruction is fetched. Since it is in offset 0x14
    of the .text segment, which itself begins at 0x8014, the
    CPU will put 0x8014 in the MAR. The memory will then
    send back the contents of that location, i.e. the
    instruction, 0x48, which the CPU will receive in the MAR.
    The instruction itself doesn’t access memory, so this is the end
    of activity in MAR/MDR for this instruction.
3.d. As pointed out in the notes, any machine code which
    the assembler had tentatively filled with an offset must
    be changed by the linker to the actual address of the
    item in question. The instruction at offset 0x27 the .text
    segment will change to 0xb900200000, because x, which
    had been at offset 0 of the .data segment, will be at
    absolute location 0x200 + 0 = 0x200. The instruction at
    offset 0x33 will change to 0x891d10200000. (The linker
    will place these two instructions at 0x8027 and 0x8033,
    but this is irrelevant to the question.)
4.a.
pushl $0
pushl $0
4.b.
movl -8(%ebp), %eax
ret

(The question was misstated. It should have asked for
two instructions, the second of which was ret. So, the
grading here was liberal.)
Note that
movl s, %eax
will NOT work. Local variables are stored on the stack,
not in the .data segment.
5.
*(&z+3) = 12;
Here is a picture of the stack at the time we are executing
g():

z saved EBP
return address
w saved EBP
v return address
u x y

As you can see, the address of w is 12 bytes more than
that of z. However, &z is considered a pointer, so pointer
arithmetic is used, hence the expression &z+3.