CMSC216: Bonus Review 2A

Chris Kauffman

Last Updated:
Sun Nov 2 09:41:43 PM EST 2025

Bonus Review Rules

- 3 Questions will be shown with about 5min per question, 15min total, time limit enforced on Gradescope Quiz
- Individual student bonus dots will be calculated as

  ```
  BonusDots = floor(log2(TotalCorrectSectionAnswers))
              - YourIncorrectAnswers
  ```
- Cooperation is allowed and encouraged within your discussion section: the more correct answers in the section, the more bonus points for all
- Staff will try to facilitate discussion but will not comment on correct/incorrect answers during the quiz
- Scores will posted after all sections have taken the done the bonus review, likely the following day
- Student in the Discussion Section with the highest `TotalCorrectSectionAnswers` will get +2 BonusDots
- Bonus Review is Open Resource just like the exam: https://www.cs.umd.edu/~profk/216/exam-rules.pdf

Staging

- Open up the Gradescope Bonus Review Quiz for the day
- Once started, the quiz closes after 15min
- Get your resources set for the quiz

Okay...

Question 1

```c
int quotient_greater(int numer,
                     int denom,
                     int thresh)
{
  int quot = numer / denom;
  if(quot > thresh){
    return 1;
  }
  else{
    return 0;
  }
}
```

```
.global quotient_greater
quotient_greater:
        movl    %edi, %eax
        cltq
        cqto
        idivl   %esi
        cmpl    %edx, %eax
        jle     .ABOVE
        movl    $1, %eax
        ret
.ABOVE:
        movl    $0, %eax
        ret
```

Above is a correct C function and a BUGGY assembly implementation. Which of the below best describes the problem in assembly and its fix?

- (A) Sign extension in preparation for the division is not correct; add a `cwtl` first.
- (B) The argument registers are used incorrectly; change the first line to `movl (%edi),%eax`
- (C) The `%esi` register cannot be used in division; copy `%esi` to a different register and use that one in division.
- (D) The argument in `%edx` is changed during division; copy `%edx` to a different register early then compare against that one later.

Question 2

Consider this struct and function prototype.

```
// packed_struct_main.c
typedef struct {
  short first;
  short second;
} twoshort_t;
```

```
short sub_struct(twoshort_t ti);
```

Which of the nearby instruction sequences will set the DX-family register to be the value of `ti.second` if placed at beginning `sub_struct()`?

```
sub_struct:
  # (A) edx = ti.second;
  movq %rdi, %rdx
  sarl $8, %edx
  andl $0xFF, %edx

  # (B) edx = ti.second;
  movl %edi, %edx
  sall $16, %edx
  andl $0xFFFF, %edx

  # (C) edx = ti.second;
  movl %edi, %edx
  sarl $16, %edx
  andl $0xFFFF, %edx

  # (D) edx = ti.second;
  movl $0, %edx
  movw 2(%rdi), %dx

  # (E) edx = ti.second;
  movzwq (%rdi), %dx

  # (F) edx = ti.second;
  xorq %rdx,%rdx
  movswq 2(%rdi), %dx
```

Question 3

```
==4012== Jump to the invalid address stated on the next line
==4012==    at 0x1320D48: ???
==4012==    by 0x4003F48: CALL_batt_update (test_batt_update_asm.s:227)
==4012==    by 0x4003733: main (test_batt_update.c:668)
==4012==  Address 0x1320d48 is not stack'd, malloc'd or (recently) free'd
==4012==
==4012== Process terminating with default action of signal 11 (SIGSEGV)
```

Which of the following is a likely cause and fix for the above error in Project 3's `batt_update()` function?

- (A) Incorrectly accessing an array location; FIX by adjusting the scaling factor on an instruction of the form
  `... (%reg1, %reg2, scale) ...`
- (B) Incorrect movement size causing data to be read or written that is out of bounds; FIX by adjusting an instruction suffix to select one of `q/l/w/b` that is appropriate to the C data type
- (C) Use of a callee-save register that is not properly restored before returning. FIX by adding a `pushq` instruction at the beginning of the function to save the register and a `popq` instruction to restore it before returning.
- (D) Failure to restore the stack pointer to its original value before returning; FIX by finding the return statement that triggered the problem and using `addq` / `popq` instructions to restore the stack to its original state.
- (E) Improper stack alignment to set up a function call. FIX by adjusting the stack pointer near the beginning of a function with a `subq` instruction to ensure 16-byte alignment accounting for the 8-byte return address on the stack already.
- (F) Incorrect syntax to access a global variable. FIX by adjusting a line like
  `movl myglobal, %eax   TO   movl myglobal(%rip), %eax`
  which uses PC-relative addressing