EECE.3170: Microprocessor Systems Design I

Summer 2016

Homework 3 Solution

1. *(25 points) Implement the following conditional statement. You may assume that "X", "Y", and "Z" refer to 16-bit variables stored in memory, which can be directly accessed using those names (for example, `MOV AX, X` would move the contents of variable "X" to the register AX). Your solution should not modify AX or BX.*

```
if (AX >= 40) {
   Z = X - Y;
}
else {
   Z = X + Y;
   if (Z > 0)
      X = BX * 8;
   else
      X = BX / 4;
}
```

Solution: Other solutions may be valid. Key points:

- Handling each conditional test appropriately (AX >= 40; Z > 0)
- Making sure your code only executes one part (if or else) of each conditional statement.
- Each mathematical operation, done without changing any required variable.

```
   MOV  DX, X                    ; Set Z = X using two MOV
   MOV  Z, DX                    ;   instructions
                                 ; Add or subtract Y later
   CMP  AX, 40                   ; Jump to else case if
   JL   else                     ;   !(AX >= 40) (if AX < 40)
   MOV  DX, Y                    ; Subtract Y from X (since
   SUB  Z, DX                    ;   Z = X before the SUB)
   JMP  done                     ; Skip else case
else:
   MOV  DX, Y                    ; Add Y to X (since Z = X
   ADD  Z, DX                    ;   before the ADD)
   MOV  X, BX                    ; Set X = BX (since X will be
                                 ;   either BX * 8 or BX / 4)
   CMP  Z, 0                     ; If Z <= 0, jump to inner
   JLE  else2                    ;   else case
   SLL  X, 3                     ; X = BX << 3 = BX * 2^3
   JMP  done                     ; Skip inner else case
else2:
   SRA  X, 2                     ; X = BX >> 2 = BX / 2^2
done:                                 ; End of code
```

2. *(25 points) Implement the following loop. As in question 1, assume "X" is a 16-bit variable in memory that can be accessed by name. (Hint: Any loop that executes the correct number of iterations is acceptable—you do not necessarily have to change your loop counter in exactly the same way as the for loop, since i is not used in the body of the loop.)*

```
for (i = 0; i < X; i++) {
   AX = AX + X;
   BX = BX - X;
   if (AX == BX)
        break;          // Exit loop early
}
```

Solution: Other solutions may be valid; the key pieces of this problem are:

- Ensuring that the assignment statements are enclosed in a loop with X iterations.
  - Note that, as mentioned above, any loop with X iterations will be valid. The solution below takes advantage of the x86 LOOP instructions so that the actual loop counts from X down to 0, rather than counting up.
- Comparing AX to BX and exiting the loop early if they are equal.
  - Note that this can be accomplished by using a LOOPNE instruction, as shown below, or by adding an explicit jump instruction that leaves the loop when the condition is true.

```
     MOV  CX, X          ; CX = X = # of loop iterations
L:   ADD  AX, X          ; AX = AX + X
     SUB  BX, X          ; BX = BX - X
     CMP  AX, BX
     LOOPNE L            ; Decrement CX, then check if
                         ;   CX is non-zero and previous compare
                         ;   result is "not equal" (AX != BX)
                         ; If either of those conditions are
                         ;   false, exit loop
```

3. *(25 points) Implement the following conditional statement. As in question 1, assume "X" and "Y" are 16-bit variables in memory that can be accessed by name. (Note: Make sure you carefully count the parentheses to make sure you combine conditions correctly!)*

```
if (((AX < X) && (BX < Y)) || ((AX > Y) && (BX > X))) {
   AX = AX - BX;
}
```

Solution: Other solutions may be possible; the key piece of this problem is the evaluation of the complex condition shown, which can be done with SETcc instructions. Note that a series of jump instructions can also be used to evaluate that condition.

```
     CMP  AX, X
     SETL DL         ; (AX < X)
     CMP  BX, Y
     SETL DH         ; (BX < Y)
     AND  DL, DH     ; ((AX < X) && (BX < Y))
     CMP  AX, Y
     SETG CL         ; (AX > Y)
     CMP  BX, X
     SETG CH         ; (BX > X)
     AND  CL, CH     ; ((AX > Y) && (BX > X))
     OR   DL, CL     ; Logical OR of previous complex conditions
                     ; DL is now 1 if the entire condition in the
                     ;   if statement is true
     JZ   SKIP       ; If result of OR is zero, skip subtraction
     SUB  AX, BX     ; AX = AX - BX
SKIP:                ; End of code
```

4. *(25 points) Implement the following loop. As in previous questions, assume "X", "Y", and "Z" are 16-bit variables in memory that can be accessed by name. Recall that a while loop is a more general type of loop than the for loop seen in question 2—a while loop simply repeats the loop body as long as the condition tested at the beginning of the loop is true.*

```
while ((Y > 0) && (X < 0)) {
   X = X + Z;
   Y = Y - X;
   Z = Z + AX;
}
```

Solution: Other solutions may be valid. The key pieces of this problem are:

- Testing the loop conditions and exiting if either one is false.
- Moving data through registers to perform the addition and subtraction operations.
- Unconditionally jumping back to the start of the loop at the end.

```
L: CMP  Y, 0            ; Exit loop if !(Y > 0) → if (Y <= 0)
   JLE  done
   CMP  X, 0            ; Exit loop if !(X < 0) → if (X >= 0)
   JGE  done
   MOV  DX, Z           ; DX = Z
   ADD  X, DX           ; X = X + DX = X + Z
   MOV  CX, X           ; CX = X
   SUB  Y, CX           ; Y = Y - CX = Y - X
   ADD  Z, AX           ; Z = Z + AX
   JMP  L               ; Return to start of loop
done:                   ; End of code
```