**Problem 1 (15 pts):** Nearby is a C function `col_update()` with associated data and documentation. **Re-implement this function in x86-64 assembly** according to the documentation given. Follow the same flow provided in the C implementation. The comments below the `colinfo_t` struct give information about how it lays out in memory and as a packed argument.

Indicate which registers correspond to which C variables.

### SOLUTION:

```assembly
.text
.globl col_update

# YOUR CODE BELOW

col_update:
    movl 0(%rdi),%esi  # cur = info->cur
    movl 4(%rdi),%edx  # step= info->step
    cmpl $0,%esi       # if(cur < 0)
    jle .ERROR
    addl $1,%edx       # step++
    testl $0x01,%esi   # if(cur%2 == 1)
    jz .EVEN          # go to even case
# ODD CASE (fall through)
    imull $3,%esi     # odd: cur *= 3
    addl $1,%esi      # odd: cur += 1
    jmp .RETURN       # jump over even

# EVEN:
    sarl $1,%esi      # even: cur /= 2

.RETURN:
    movl %esi,0(%rdi)  # info->cur = cur;
    movl %edx,4(%rdi)  # info->step= step;
    movl $0,%eax       # success
    ret                # return 0

.ERROR:
    movl $1,%eax       # error case
    ret                # return 1
```

```c
typedef struct{
  int cur;
  int step;
} colinfo_t;

int col_update(colinfo_t *info){
  // Updates current value and step in
  // colinfo_t pointed by param info. If
  // infor->cur is invalid, makes no changes
  // and returns 1 to indicate an
  // error. Otherwise performs odd or even
  // update on cur and increments step
  // returning 0 for success.
  int cur = info->cur;
  int step = info->step;
  if(cur <= 0){
    return 1;
  }
  step++;  
  if(cur % 2 == 1){
    cur = cur*3+1;
  }else{
    cur = cur / 2;
  }
  info->cur = cur;
  info->step = step;
  return 0;
}
```

### Problem Description:

- **Problem Statement:** A C function `col_update()` with associated data and documentation. The task is to re-implement this function in x86-64 assembly according to the documentation given.
- **Context:** The comments below the `colinfo_t` struct provide information about how it lays out in memory and as a packed argument.
- **Task:** Indicate which registers correspond to which C variables.

### Solution:

- **Assembly Code:**
  ```assembly
  .text
  .globl col_update

  # YOUR CODE BELOW

  col_update:
  movl 0(%rdi),%esi  # cur = info->cur
  movl 4(%rdi),%edx  # step= info->step
  cmpl $0,%esi       # if(cur < 0)
  jle .ERROR
  addl $1,%edx       # step++
  testl $0x01,%esi   # if(cur%2 == 1)
  jz .EVEN          # go to even case
  # ODD CASE (fall through)
  imull $3,%esi     # odd: cur *= 3
  addl $1,%esi      # odd: cur += 1
  jmp .RETURN       # jump over even

  # EVEN:
  sarl $1,%esi      # even: cur /= 2

  .RETURN:
  movl %esi,0(%rdi)  # info->cur = cur;
  movl %edx,4(%rdi)  # info->step= step;
  movl $0,%eax       # success
  ret                # return 0

  .ERROR:
  movl $1,%eax       # error case
  ret                # return 1
  ```

- **C Code:**
  ```c
  typedef struct{
    int cur;
    int step;
  } colinfo_t;

  int col_update(colinfo_t *info){
    // Updates current value and step in
    // colinfo_t pointed by param info. If
    // infor->cur is invalid, makes no changes
    // and returns 1 to indicate an
    // error. Otherwise performs odd or even
    // update on cur and increments step
    // returning 0 for success.
    int cur = info->cur;
    int step = info->step;
    if(cur <= 0){
      return 1;
    }else{
      step++;
      if(cur % 2 == 1){
        cur = cur*3+1;
      }
      else{
        cur = cur / 2;
      }
      info->cur = cur;
      info->step = step;
      return 0;
    }
  }
  ```

### Reflection:

- The solution matches the requirements by accurately implementing the specified function in assembly, adhering to the flow provided in the C implementation.
- The correspondence between C variables and assembly registers is indicated in the solution.

---

1/2
**Problem 2 (15 pts):** Below is an initial register/memory configuration along with snippets of assembly code. Each snippet is followed by a blank register/memory configuration which should be filled in with the values to reflect changes made by the preceding assembly. The code is continuous so that POS A is followed by POS B.

```
SOLUTION:  
addl %edi, %esi  
subq $8, %rsp  
movl $100, 4(%rsp)  
movl $300, 0(%rsp)  
addl (%rsp), %eax  
addl (%rdi), %rax  

INITIAL  
|-------+-------|  |-------+-------|  |-------+-------|  
| REG | Value |  | REG | Value |  | REG | Value |
|-------+-------|  |-------+-------|  |-------+-------|  
| rax | 10 |  | rax | 10 |  | rax | 10 |
| rdi | 20 |  | rdi | 20 |  | rdi | 20 |
| rsi | 30 |  | rsi | 30 |  | rsi | 30 |
| rsp | #3032 |  | rsp | #3032 |  | rsp | #3032 |
|-------+-------|  |-------+-------|  |-------+-------|  
| MEM | Value |  | MEM | Value |  | MEM | Value |
|-------+-------|  |-------+-------|  |-------+-------|  
| #3032 | 250 |  | #3032 | 250 |  | #3032 | 250 |
| #3028 | 1 |  | #3028 | 1 |  | #3028 | 1 |
| #3024 | 2 |  | #3024 | 2 |  | #3024 | 2 |
| #3020 | 3 |  | #3020 | 3 |  | #3020 | 3 |
```

**Problem 3 (10 pts):** Rover Witer is writing an assembly function called `compval` which he will use in C programs. He writes a short C `main()` function to test `compval` but is shocked by the results which seem to defy the C and assembly code. Valgrind provides no insight for him. Identify why Rover’s code is behaving so strangely and fix `compval` so it behaves correctly.

**Sample Compile / Run:**
```
> gcc compval_main.c compval_asm.s  
> a.out  
expect: 0  
actual: 19  
expect: 0  
actual: 50
```

SOLUTION: The `movq` instruction at line 7 of `compval` writes 8 bytes. This is inappropriate as a 4-byte int is supposed to be written. Apparently the stack layout in `main()` has the variable `actual` at a memory address immediately below variable `expect` so that on writing 8 bytes, the low order 4 bytes correctly get written to `actual` but the high order 4 bytes (all 0’s for small values) overwrite the variable `expect` leaving it as 0. The fix for this is to use `movl %eax, (%rdx)` which will write 4 bytes, filling only actual.