Yingjie Lian
HW- 3
CS-3810
02.05.2018

Answer 1:

The **Bold** comments are my comments for the assembly codes.

new-proc: **# new-proc function**
sll $a0, $a0, 24 **# shifts a0 left by 24**
srl $a0, $a0, 24 **# shifts a0 right by 24**
add $v0, $a0, $zero **# set's $a0 to $v0 i.e, $v0=$a0**
jr $ra **# returns from this function**

In one sentence, **# this function set's $v0 with first byte of $a0**

Answer 2:

The bold words are my comments for this assembly language codes

Note: $zero is a register that holds constant value 0

new-proc: **// New procedure**
blt $a1, $zero, loop2 **// blt is branch if less than ; branch to loop2 if a1<zero**
loop 1: **//loop1 begins**
beq $a1, $zero, proc-end **// beq is branch if equal ; branch to proc-end if a1 is equal to zero**
sll $a0, $a0, 1 **// sll is shift left logical ; shift a0 left by 1 place and store it back in a0**
addi $a1, $a1, -1 **// addi is bitwise and register with immediate value and store result in a1**
j loop1 **// jumps to loop1**
loop2: **// loop2 starts**
beq $a1, $zero, proc-end **// if a1 is equal to zero branch to proc-end**
srl $a0, $a0, 1 **// srl is shift right logical ; shift a0 right by 1 place and store result in a0**
addi $a1, $a1, 1 **// bitwise and a1 with 1 and store result in a1**
j loop2 **// jump to loop2 . It is unconditional jump**

proc-end: **// proc-end begins**

add $v0, $a0, $zero **// add 2 register i.e. a0 and zero ,and store the result in v0**

jr $ra **// jump to register ra**

Simple equation:

Since here zero =0

```
if(a1<zero)
{
     while(a1!=zero)
      {
       a0= a0/2;
       a1=a1+1;
     }
   v0= a0+zero;
}
else
{
     while(a1!=zero){
          a0=a0*2;
       a1 = a1 -1;
       }
   v0= a0+zero;
}
```

So,equation becomes:

a0 = a0/(2^|a1|) for a1 < 0

a0 = a0 * 2^|a1| for a1 >= 0

v0 = a0

Answer 3:

Based on the class notes, we know that X, Y, P, and Q are:

X:

```
$sp = $sp - 16     # create space for 4 values on the stack.
                   # since the stack grows from a high address to a
                          # low address, an increase in stack size corresponds
                          # to a decrease in the stack pointer value
  sw $a0, 12($sp)    # store the result in $a0 into the memory address
                   # indicated by $sp+16
  sw $ra, 8($sp)     # save the second value on stack
  sw $t0, 4($sp)     # save the third value on stack
  sw $fp, 0($sp)     # save the fourth value on stack
  $fp = $sp          # set the frame pointer to the stack pointer

Y:
lw $fp, 0($sp)     # start restoring values from stack
  lw $t0, 4($sp)
  lw $ra, 8($sp)
  lw $a0, 12($sp)
  $sp = $sp + 16     # decrement the size of the stack

P:
$sp = $sp - 4      # create space on the stack for one save
  sw $s0, 0($sp)     # save the value in $s0

Q:
lw $s0, 0($sp)     # restore the value of $s0 from the stack
  $sp = $sp + 4      # decrement the stack size
```