۲)

```
L۲:  addi  sp , sp , -8   ⇒  تعریف ۲ تا ۴ بایت برای رجیستر stack painter
     sw    a۰ , ۴ (sp)    ⇒  store word ⇒ برای ذخیره رجیستر a۰ و ra (return address)
     sw    ra , ۰ (sp)
     addi  t۰ , zero, ۱   ⇒  ذخیرهٔ یک Temporary و برابر قرار دادن آن با ۱
     blt   t۰ , a۰ , L۱   ⇒  دستور blt در Risc ۵ سود و اینستراکشن bgt هست یعنی اگر مقدار n بزرگتر از ۱ باشد، به برچسب L۱ برو!
     addi  a۰ , zero , ۱  ⇒  return ۱   ← این دستور می‌گوید اگر شرط برقرار نبود
     addi  sp , sp , ۸    ⇒  restore sp
     jalr  zero , ra , ۰  ⇒  معادل jr ra هست؛ یعنی برگرد کن!

L۱:  addi  a۰ , a۰ , -۱   ⇒  معادل n=n-۱ هست؛ a۰ را با immediate -۱ جمع میکند.
     jal   ra , L۲        ⇒  به صورت recursive function به بالا L۲ (برچسب) می‌رود ← خروجی در ra
     lw    t۱ , ۴ (sp)    ⇒  ra , ta رو از stack می‌خواند!
     lw    ra , ۴ (sp)
     addi  sp , sp , ۸    ⇒  restore sp
     mul   a۰ , t۱ , a۰   a۰ = n * factorial (n-۱) ⇒  حاصلضرب را در a۰ می‌ریزد.
     jalr  zero, ra , ۰   کلاً برگشت می‌کند.
```

ب) برنامهٔ فوق برنامه‌ای است که تابع فاکتوریل را محاسبه می‌کند به صورت بازگشتی و ترجمه شدهٔ برنامهٔ C زیر است

```
int factorial (int n){
    if (n <= 1)
        return 1;
    else
        return (n * factorial(n-1));
```

recursive function ... خروجی در

۲-ج)

کد سی :

```
int factorial (int n) {
    int result = 1;
    While (n > 1) {
        result *= n;
        n --; }
    return result; }
```

بازنویسی

```
addi   a0, a0, 0        # a0 = n  => ورودی
addi   s0, zero, 1      # s0 = result = 1
addi   t0, zero, 0      # t0 = 0
while:  beq a0, t0, done
        mul s0, a0, s0   => result = result * n
        addi a0, a0, -1  => n = n-1
        j   while        # repeat the loop
done:
```