Assembly Code Analysis: C to ARM Mapping and Error Analysis

1. C Code to Assembly Mapping

Register Usage Mapping

- **x19**: `count` parameter (second function argument)
- **x20**: `len` (string length)
- **x21**: `str` parameter (first function argument)
- **x22**: `out` (char** array)
- **x23**: `current` (temporary string buffer)
- **x24**: `current_len` (length of current string)
- **x25**: loop counter `i`
- **x26**: pointer for array indexing

C Code to Ground Truth (gd) Assembly Mapping

c

```c
int len = strlen(str);
```

asm

```asm
mov x21, x0          ; store str parameter
bl  _strlen          ; call strlen
mov x20, x0          ; store len = strlen result
```

c

```c
char **out = malloc(len * sizeof(char *));
```

asm

```asm
sbfiz x0, x0, #3, #32  ; multiply len by 8 (sizeof(char*))
bl  _malloc          ; allocate memory
mov x22, x0          ; store out pointer
```

c

```c
char *current = malloc(len + 1);
current[0] = '\0';
```

asm

```
mov x8, #4294967296  ; prepare len+1 calculation
add x8, x8, x20, lsl #32
asr x0, x8, #32      ; x0 = len + 1
bl  _malloc          ; allocate memory
mov x23, x0          ; store current pointer
strb  wzr, [x0]       ; current[0] = '\0'
```

c

```
for (int i = 0; i < len; ++i) {
```

asm

```
cmp w20, #1          ; check if len >= 1
b.lt  LBB0_3          ; skip loop if len < 1
and x25, x20, #0xffffffff ; initialize loop counter
```

c

```
size_t current_len = strlen(current);
```

asm

```
mov x0, x23          ; pass current to strlen
bl  _strlen          ; call strlen
mov x24, x0          ; store current_len
```

c

```
current = realloc(current, current_len + 2);
```

asm

```
add x1, x0, #2       ; current_len + 2
mov x0, x23          ; pass current pointer
bl  _realloc         ; reallocate memory
mov x23, x0          ; update current pointer
```

c

```
current[current_len] = str[i];
current[current_len + 1] = '\0';
```

asm

```
ldrb  w8, [x21], #1   ; load str[i] and increment str pointer
add x9, x0, x24     ; calculate &current[current_len]
strb  w8, [x9]        ; current[current_len] = str[i]
strb  wzr, [x9, #1]   ; current[current_len + 1] = '\0'
```

c

```
out[i] = malloc(strlen(current) + 1);
strcpy(out[i], current);
```

asm

```
bl  _strlen          ; strlen(current)
add x0, x0, #1       ; strlen + 1
bl  _malloc          ; allocate memory
str x0, [x26], #8    ; store in out[i] and increment pointer
mov x1, x23          ; source = current
bl  _strcpy          ; copy string
```

2. Vertical Comparison: Ground Truth vs Predicted

Differences Found:

Difference 1: Memory allocation calculation

Ground Truth (gd):

asm

```
sbfiz x0, x0, #3, #32  ; multiply by 8 using shift+sign extend
```

Predicted (pred):

asm

```
lsl x23, x0, #32     ; shift left by 32 bits
asr x0, x23, #29     ; arithmetic shift right by 29 bits
```

Difference 2: String character appending

Ground Truth (gd):

```asm
ldrb  w8, [x21], #1   ; load str[i], increment pointer
add x9, x0, x24     ; calculate address: current + current_len
strb  w8, [x9]        ; store character at correct position
strb  wzr, [x9, #1]   ; null terminate at next position
```

Predicted (pred):

```asm
ldrb  w8, [x21], #1   ; load str[i], increment pointer
strb  w8, [x0, x0]    ; ✗ CRITICAL ERROR: uses x0 as both base and offset
strb  wzr, [x0, #1]   ; null terminate at wrong position
```

Difference 3: strlen call position

Ground Truth (gd):

```asm
mov x0, x23         ; set up parameter for strlen
bl  _strlen         ; call strlen
mov x24, x0         ; save current_len before realloc
add x1, x0, #2      ; calculate new size
```

Predicted (pred):

```asm
add x1, x0, #2      ; calculate new size first
mov x0, x23         ; then set up realloc parameters
bl  _realloc        ; call realloc
; ✗ Missing: saving current_len in x24
```

3. Error Analysis and Root Causes

Error 1: Incorrect Memory Allocation Size

Location: Initial `out` array allocation **C Code**: `char **out = malloc(len * sizeof(char *));`

Problem: The predicted code uses `lsl x23, x0, #32` followed by `asr x0, x23, #29`, which effectively multiplies by 8 but through a convoluted 32-bit shift operation that may not handle large values correctly.

Root Cause: This appears to be a translation error from x86 assembly where the original code used:

```asm
shlq  $32, %rbx        ; shift left 32 bits
sarq  $29, %rdi        ; shift right 29 bits
```

The translator incorrectly preserved this x86 idiom instead of using ARM's more direct `sbfiz` instruction.

Error 2: Critical String Building Bug

Location: Character appending in loop **C Code**: `current[current_len] = str[i];`

Problem: The predicted code uses `strb w8, [x0, x0]` which means it's storing the character at address `current + current`, but `x0` contains the `current` pointer, not the `current_len`. This will write to incorrect memory locations.

Root Cause: The translator failed to properly track that `x24` should contain `current_len` for the indexing operation. In the x86 version, `%r14` held `current_len`, but the ARM translation lost this mapping.

Error 3: Missing Variable Preservation

Location: Before realloc call **C Code**: `current = realloc(current, current_len + 2);`

Problem: The predicted code doesn't save `current_len` in `x24` before the realloc call, so when it tries to index with `current_len`, the value is lost.

Root Cause: This is an optimization error where the translator assumed it could reuse registers without preserving intermediate values that are needed after function calls.

Impact of Errors

1. **Error 1** may cause incorrect memory allocation sizes for large strings
2. **Error 2** will cause memory corruption and incorrect string building
3. **Error 3** will cause use of uninitialized/incorrect offset values

The predicted assembly will likely crash or produce completely incorrect results due to the string indexing bug (Error 2), making it functionally broken compared to the original C code intent.