Problem 18 Assembly Analysis: Music String Parser

1. C Code to Assembly Mapping

Register Usage Mapping

- **x19**: `count` parameter (output parameter)
- **x20**: `temp_music_string` (allocated string buffer)
- **x21**: `out` (dynamic array pointer)
- **x22**: `music_string_length` (string length + 1)
- **x23**: loop counter `i`
- **x24**: `size` (current array size)
- **x25**: `capacity` (array capacity)
- **x26**: constant 4 (initial capacity)
- **x27**: constant 31855 (hash value for "o|")
- **x28**: constant values (varies between gd and pred)
- **Stack offsets**: `current` buffer stored at `[sp, #12]` or `[sp, #14]`

C Code to Ground Truth (gd) Assembly Mapping

c

```c
int music_string_length = strlen(music_string) + 1;
char *temp_music_string = malloc(music_string_length + 1);
strcpy(temp_music_string, music_string);
strcat(temp_music_string, " ");
```

asm

```asm
mov x21, x0                    ; save music_string
bl  _strlen                    ; get string length
mov x22, x0                    ; save length
add w8, w22, #2                ; length + 2 for space and null
sxtw  x0, w8                    ; sign extend to 64-bit
bl  _malloc                    ; allocate memory
mov x26, x0                    ; save temp_music_string
mov x1, x21                    ; source = music_string
bl  _strcpy                    ; copy string
bl  _strlen                    ; get length of copied string
mov w8, #32                    ; ASCII space character
strh  w8, [x26, x0]             ; append space at end
```

c

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

asm

```asm
add w22, w22, #1                ; increment length for loop
mov x20, x26                    ; set pointer to temp string
b LBB0_5                       ; jump to loop condition
```

c

```c
if (temp_music_string[i] == ' ') {
```

asm

```asm
ldrb  w19, [x20]                ; load current character
cmp w19, #32                   ; compare with space (32)
b.ne  LBB0_17                   ; if not space, go to else branch
```

c

```c
if (strcmp(current, "o") == 0) {
    out[size++] = 4;
}
```

asm

```asm
ldrh  w8, [sp, #12]             ; load current buffer (2 bytes)
cmp w8, #111                   ; compare with 'o' (111)
b.ne  LBB0_10                   ; skip if not equal
; ... capacity check and realloc ...
str w24, [x21, w23, sxtw #2]    ; store 4 in out[size]
add w23, w23, #1                ; increment size
```

c

```c
if (strcmp(current, "o|") == 0) {
    out[size++] = 2;
}
```

```asm
ldrh  w8, [sp, #12]               ; load current buffer
eor w8, w8, w25                  ; XOR with 31855 ("o|" hash)
ldrb  w9, [sp, #14]               ; load overflow byte
orr w8, w8, w9                   ; combine with overflow
cbnz  w8, LBB0_14                 ; skip if not "o|"
; ... capacity check and realloc ...
mov w8, #2                       ; value to store
str w8, [x21, w23, sxtw #2]     ; store 2 in out[size]
add w23, w23, #1                 ; increment size
```

2. Vertical Comparison: Ground Truth vs Predicted

Major Structural Differences:

Difference 1: Stack Layout

Ground Truth (gd):

```asm
strb  wzr, [sp, #14]              ; initialize overflow byte
strh  wzr, [sp, #12]              ; initialize current buffer
```

Predicted (pred):

```asm
strb  wzr, [sp, #15]              ; different stack offset
strh  wzr, [sp, #14]              ; different stack offset
```

Difference 2: Loop Structure

Ground Truth (gd):

```asm
add x20, x20, #1                 ; increment string pointer
subs  x22, x22, #1                ; decrement counter
b.eq  LBB0_20                     ; exit if zero
```

Predicted (pred):

```asm
add x23, x23, #1                ; increment index
cmp x22, x23                    ; compare with length
b.eq  LBB0_21                    ; exit if equal
```

Difference 3: Register Usage for Array Operations

Ground Truth (gd):

```asm
str w8, [x21, w23, sxtw #2]     ; correct array indexing
```

Predicted (pred):

```asm
str w26, [x21, w24, sxtw #2]    ; uses different register
```

Difference 4: Realloc Parameter Setup

Ground Truth (gd):

```asm
mov x0, x21                     ; pass current array pointer
bl  _realloc                    ; call realloc
mov x21, x0                     ; update array pointer
```

Predicted (pred):

```asm
bl  _realloc                    ; ✖ missing x0 parameter setup
```

Difference 5: String Comparison Constants

Ground Truth (gd):

```asm
mov w27, #31790                 ; hash for ".|"
```

Predicted (pred):

```asm
mov w28, #2                     ; ✖ wrong constant
```

Difference 6: Control Flow Error

Predicted (pred) has extra unreachable code:

```asm
LBB0_19:                        ; unreachable block
  add w24, w24, #1
  mov w8, #1
  str w8, [x21, w24, sxtw #2]
LBB0_20:                        ; infinite loop
  strb  wzr, [sp, #14]
LBB0_21:
  add x23, x23, #1
  cmp x22, x23
  b.ne  LBB0_20                 ; ✖ branches back to LBB0_20
```

3. Error Analysis and Root Causes

Error 1: Missing Realloc Parameter

Location: Multiple realloc calls **C Code**: `out = realloc(out, capacity * sizeof(int));`

Problem: The predicted code calls `bl _realloc` without setting up `x0` with the current array pointer.

Root Cause: The translator failed to track that the x86 code sets up `%rdi` before calling realloc, and didn't generate the corresponding `mov x0, x21` instruction in ARM.

Error 2: Incorrect String Comparison

Location: String pattern matching **C Code**: `if (strcmp(current, ".|") == 0)`

Problem: The predicted code uses `mov w28, #2` instead of the correct hash value `mov w27, #31790` for comparing ".|".

Root Cause: The translator incorrectly mapped the x86 immediate values used for string comparison, losing the actual hash constants.

Error 3: Control Flow Corruption

Location: Main loop structure **C Code**: `for (int i = 0; i < music_string_length; i++)`

Problem: The predicted code creates an infinite loop between `LBB0_20` and `LBB0_21` due to incorrect branch targets.

Root Cause: The translator failed to properly map the x86 jump instructions to ARM branches, creating unreachable code and incorrect loop termination.

Error 4: Register Confusion

Location: Array indexing operations **C Code**: `out[size++] = value;`

Problem: The predicted code uses inconsistent registers for array operations, mixing `w24` and `w25` incorrectly.

Root Cause: The translator didn't maintain consistent register mappings from x86 to ARM, causing confusion between size/capacity variables.

Error 5: Stack Offset Misalignment

Location: Local variable access **C Code**: `char current[3] = "";`

Problem: The predicted code uses different stack offsets (`sp, #15` vs `sp, #14`) which could cause memory corruption.

Root Cause: The translator didn't properly calculate ARM stack frame layout, leading to misaligned local variable access.

Impact Assessment

1. **Error 1**: Will cause segmentation faults due to invalid realloc calls
2. **Error 2**: Will fail to recognize ".|" patterns, producing incorrect output
3. **Error 3**: Will cause infinite loops, making the program hang
4. **Error 4**: Will corrupt array data due to incorrect indexing
5. **Error 5**: Could cause stack corruption and unpredictable behavior

The predicted assembly is completely non-functional due to the combination of these errors, particularly the control flow corruption and missing realloc parameters.