Project 1
EGR 424 - The Design of Microcontroller Applications
Instructor: Dr. Chirag Parikh
Renzo Garza Motta
September 30th, 2022

Objectives

The objective of this project is to develop a familiarity with ARM assembly language and the ARm development tool suite. This familiarity is demonstrated by studying the assembly representation of a C function.

Function

The function explored in this project is the stpncpy.c function in the C language. This function takes in three arguments, dst, src, and count. The first and second arguments, char *dst (destination) and char *src (source), represents pointers to an address in memory which is expected to have been allocated for strings. The last argument, count, is of type size_t and represents the amount of characters that are to be transferred from one string to another.

In action, stpncpy() is given a source string from which a certain number of letters (equal to the passed argument count) are copied over to the destination string. Additionally, the function returns the address of the last character in the destination string.

Compilation

The provided source file was compiled using the previously outlined alias CC. This alias is defined as

```
alias CC='arm-none-eabi-gcc -Wall -O3 -march=armv7-m -mcpu=cortex-m3 -mthumb -mfix-cortex-m3-ldrd'
```

Further, given the source file, the command

```
CC -DPREFER_SIZE_OVER_SPEED -Os -S stpncpy.c
```

Was used to compile and get an assembly output. An option was provided when compiling these functions. Optimize for speed (-O3) or optimize for size (-Os). These optimizations are as they outlined one is less memory and the other is faster. However, with these optimizations, other drawbacks are presented. This project calls to choose for whichever optimization yields the least amount of code.

In this case, the optimization that yielded the least amount of code was the size-over-speed (-Os) optimization with only ~31 instructions (not considering directives and other assembly syntax). Conversely, the speed-over-size optimization yields a rather drastic difference. This optimization outputs 70+ instructions.

Summary

Below, a 1-page maximum summary per instruction is outlined from the compiled code for stpncpy. Although assembly instructions are pretty simple when isolated, they can be overlooked when considering their complexity when in conjunction with one another.

Each of these summaries includes a brief description of what the instruction does, along with what label to find the instruction under within the assembly code. Additionally, a list of relevant registers, stack memory, and conditional flags (NZCV) can be found with each instruction to follow the memory changes, and operations being performed.

Each of these summaries includes a highlighted cell within the tables that indicates the register affected by the instruction at hand. Also, consider that when a CMP (comparison) instruction is executed, the register(s) being compared are highlighted in yellow.

These summaries were also executed under the assumption all registers have an initial value of 0 and that when called, the compiler locates the passed arguments *dst, *src, and count in R0, R1, and R2, respectively. Likewise, when the function returns a value, this will be stored in the R0 register.

When analyzing the assembly code, a decision tree was used to help follow the memory transfer between registers.

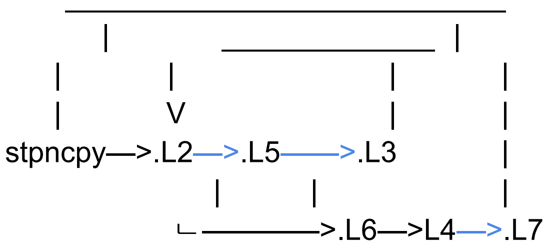

Figure 1. Decision Tree Diagram

Lastly, some test inputs were used to feed through the assembly code as an attempt to understand the logic behind some of the decisions made by the compiler. This approach provided some changing variables instead of abstract expressions to be able to more easily follow the memory trail. This can be found in the appendix.

Timing Analysis

In addition to the instructions analysis, a time analysis is required to determine how long each instruction will take to get a cumulative approximation of how well this function performs.

Table 1. Time Analysis Summary

Time Analysis						
Mnemonic	Expression	Variables				Clock Cycles
		P	**B**	**N**	**W**	
PUSH	1+N	-	-	3	-	4
MOV	1	-	-	-	-	1
MOV	1	-	-	-	-	1
B	1+P	1	-	-	-	2
LDRB	2	-	-	-	-	2
SUBS	1	-	-	-	-	1
STRB	2	-	-	-	-	2
ADDS	1	-	-	-	-	1
ADD	1	-	-	-	-	1
CBNZ	1	-	-	-	-	1
MOV	1	-	-	-	-	1
MOV	1	-	-	-	-	1
MOV	1	-	-	-	-	1
MOVS	1	-	-	-	-	1
B	1+P	1	-	-	-	2
MOV	1	-	-	-	-	1
CMP	1	-	-	-	-	1
BNE	1	-	-	-	-	1
MOV	1	-	-	-	-	1
B	1+P	1	-	-	-	2
MOV	1	-	-	-	-	1
STRB	2	-	-	-	-	2
SUBS	1	-	-	-	-	1
ADDS	1	-	-	-	-	1
CMP	1	-	-	-	-	1
BNE	1	-	-	-	-	1
ADDS	1	-	-	-	-	1
CMP	1	-	-	-	-	1
IT	1^e	-	-	-	-	1
MOVEQ	1	-	-	-	-	1
POP	1+N+P	1	-	3	-	5
					Total	**44**

As outlined by Table 1, the function stpncpy take approximately 44 clock cycles to execute. However, that value is flexible given that a clock cycle is dependent on the frequency at which the Microcontroller is being run at. The MSP432 runs by default on 3 MHz, which means that the period of a single clock cycle is $T\ =\ \frac{1}{f} = \frac{1}{3*10^{6}} =\ 0.333\ \frac{ms}{clock\ cycle}$. With this relationship outlined, 44 clock cycles takes approximately $44\ clk\ \ *\ \ 0.333\frac{ms}{clk} =\ 14.652\ ms$. This period of time, however, would change if the MSP432 was to run at its full potential of 48MHz. At this speed, the period length would be $T\ =\ \frac{1}{f} = \frac{1}{48*10^{6}} =\ 0.0208\ \frac{ms}{clock\ cycle}$ and the function would run in as little as $44\ clk\ \ *\ \ 0.0208\frac{ms}{clk} =\ 0.92\ ms$.

Instructions Summary

Instruction #1
Label: stpncpy:
Line: 25
Instruction: **PUSH** `{R4, R5, LR}`
Comments: Push registers R4, R5, and LR (Link Register) onto the Stack memory. Given that the stpncpy has three inputs (*dst, *src, and count) and a return value, the return value is assumed to be stored into R0, while the three parameters are stored in R0, R1, and R2. Since more operations could be performed, as a safety measure, the values of R4 and R5 are pushed onto the stack to prevent data loss. Additionally, the address which the program is to return to after the function is finished is stored in the LR which is also moved to the stack in case that subroutine is called.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count	0	0	0	0	0x1000
After	*dst[0]	*src[0]	count	0	0	0	0	0x0988

Stack Diagram

Address	Before	After
0x1000	0	LR
0x0996	0	R5
0x0992	0	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	0	0	0	0

Instruction #2
Label: stpncpy:
Line: 26
Instruction: `MOV     R3, R0`
Comments: Move the *dst (address) from R0 to R3 to begin performing operations and memory allocation.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count	0	0	0	0	0x0988
After	*dst[0]	*src[0]	count	*dst[0]	0	0	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	0	0	0	0

Instruction #3
Label: stpncpy:
Line: 27
Instruction: `MOV IP, #0`
Comments: Change the value of the IP register to 0 to use as a counter.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count	*dst[0]	0	0	0	0x0988
After	*dst[0]	*src[0]	count	*dst[0]	0	0	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	0	0	0	0

Instruction #4
Label: stpncpy:
Line: 28
Instruction: `B .L2`
Comments: Branch to label .L2

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count	*dst[0]	0	0	0	0x0988
After	*dst[0]	*src[0]	count	*dst[0]	0	0	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	0	0	0	0

Instruction #5
Label: .L5
Line: 30
Instruction: `LDRB    R5, [R1, IP]`
Comments: Load the value stored at the address of *src with an offset of IP and store this into R5.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count	*dst[0]	0	0	0	0x0988
After	*dst[0]	*src[0]	count	*dst[0]	0	*src[0 + 0]	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	0	0	0	0

Instruction #6
Label: .L5
Line: 31
Instruction: **`SUBS`** `R2, R2, #1`
Comments: Decrement count by 1 to prepare for data transfer between source and destination.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count	*dst[0]	0	*src[0 + 0]	0	0x0988
After	*dst[0]	*src[0]	count–	*dst[0]	0	*src[0]	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #7

Label: .L5
Line: 32
Instruction: `STRB    R5, [R0, IP]`
Comments: Copy the value of *scr at index PI into *dst at index PI

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0]	*src[0]	count–	*dst[0]	0	*src[0]	0	0x0988
After	*dst[0] = *src[0]	*src[0]	count	*dst[0]	0	*src[0]	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	0	1 or 0	0	0

Instruction #8

Label: .L5
Line: 33
Instruction: **ADDS** `R4, R3, #1`
Comments: Increment *dst[0] by 1 to access the next area of memory in the array.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0] = *src[0]	*src[0]	count	*dst[0]	0	*src[0]	0	0x0988
After	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[0 + 1]	*src[0]	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	1 or 0	0	0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #9

Label: .L5
Line: 34
Instruction: `ADD     IP, IP, #1`
Comments: Increment the counter IP to access the next memory location in the strings.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[0 + 1]	*src[0]	0	0x0988
After	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[1]	*src[0]	1	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #10

Label: .L5
Line: 35
Instruction: **CBNZ** `R5, .L3`
Comments: Determine if the current *src character is a NULL character. If the character is not a NULL character, branch to .L3.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[1]	*src[0]	1	0x0988
After	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[1]	*src[0]	1	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #11

Label: .L5
Line: 36
Instruction: `MOV     R0, R3`
Comments: If the current character in *src is not NULL, update R0 with *dst[0].

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[1]	*src[0]	1	0x0988
After	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[1]	*src[0]	1	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #12

Label: .L5:
Line: 37
Instruction: **MOV** R3, R4
Comments: Move the *dst (address) from R4 to R3.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[0] = *src[0]	*src[0]	count	*dst[0]	*dst[1]	*src[0]	1	0x0988
After	*dst[0] = *src[0]	*src[0]	count	*dst[1]	*dst[1]	*src[0]	1	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #13

Label: .L6
Line: 39
Instruction: `MOV     R4, R2`
Comments: Move count variable to R4.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988
After	0	*src[0]	0	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	1	0	0
After	0	1	0	0

Instruction #14

Label: .L6
Line: 40
Instruction: `MOVS    R1, #0`
Comments: Clear the *src address.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	*src[0]	0	*dst[2]	0	*src[1]	2	0x0988
After	0	0	0	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	1	0	0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #15

Label: .L6
Line: 41
Instruction: `B .L4`
Comments: Branch to .L4.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	2	0x0988
After	0	0	0	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #16

Label: .L3:
Line: 43
Instruction: **MOV** R3, R4
Comments: Move the *dst (address) from R4 to R3.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[1] = *src[1]	*src[0]	0	*dst[1]	*dst[2]	*src[1]	2	*dst[1] = *src[1]
After	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	*dst[1] = *src[1]

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #17

Label: .L2:
Line: 45
Instruction: `CMP     R2, #0`
Comments: Check if count is zero.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988
After	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	0	1 or 0	0	0

Instruction #18

Label: .L2:

Line: 46

Instruction: **BNE** .L5

Comments: If count is not zero, branch to .L5

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988
After	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	1 or 0	0	0
After	0	0	0	0

Instruction #19

Label: .L2:
Line: 47
Instruction: **MOV** `R0, R2`
Comments: If count is zero, store count (R2) in R0.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988
After	0	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	0	0	0
After	0	1	0	0

Instruction #20

Label: .L2:
Line: 48
Instruction: `B .L6`
Comments: Branch to .L6

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988
After	0	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	0	1	0	0
After	0	1	0	0

Instruction #21

Label: .L7:
Line: 50
Instruction: MOV IP, #0
Comments: Reset the IP counter for the program.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	2	0x0988
After	0	0	0	*dst[2]	0	*src[1]	0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	0	1 or 0	1 or 0
After	1 or 0	0	1 or 0	1 or 0

Instruction #22

Label: .L7:
Line: 51
Instruction: **STRB** `IP, [R3, R1]`
Comments: Store the value in the IP register into the address of R3 shifted by an offset of R1.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	0	0x0988
After	0	0	0	*dst[2]	0	*src[1]	*dst[2 + 0] = 0	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	0	1 or 0	1 or 0
After	1 or 0	0	1 or 0	1 or 0

Instruction #23

Label: .L7:
Line: 52
Instruction: `SUBS    R4, R4, #1`
Comments: Subtract 1 from the current value of R4 and store it back into the same register. Force conditional flag update.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	*dst[2 + 0]	0x0988
After	0	0	0	*dst[2]	-1	*src[1]	*dst[2 + 0]	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1	1 or 0	1 or 0	1 or 0

Instruction #24

Label: .L7:

Line: 53

Instruction: `ADDS    R1, R1, #1`

Comments: Add 1 to the current value in register R1. Force conditional flag update.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	-1	*src[1]	*dst[2 + 0]	0x0988
After	0	1	0	*dst[1]	-1	*src[1]	*dst[2 + 0]	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Label: .L4:
Line: 55
Instruction: **CMP** `R4, #0`
Comments: Check if the count is equal to zero

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	2	0x0988
After	0	0	0	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Label: .L4:
Line: 56
Instruction: **BNE** .L7
Comments: If count is not equal to zero, branch to .L7

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	2	0x0988
After	0	0	0	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	0	1 or 0	1 or 0

Instruction #27

Label: .L4:
Line: 57
Instruction: `ADDS    R2, R3, R2`
Comments: Add the values in R3 and R2 and store them in R2

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	0	*dst[2]	0	*src[1]	2	0x0988
After	0	0	*dst[0 +2]	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	0	1 or 0	1 or 0
After	1 or 0	1	1 or 0	1 or 0

Instruction #28

Label: .L4:
Line: 58
Instruction: `CMP     R0, #0`
Comments: Check if the value at R0 is zero.

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988
After	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1	1 or 0	1 or 0
After	1 or 0	1 or 0	1 or 0	1 or 0

Instruction #29

Label: .L4:
Line: 59
Instruction: **IT** EQ
Comments: Execute IT Block if R0 = 0

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988
After	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1 or 0	1 or 0	1 or 0
After	1 or 0	1	1 or 0	1 or 0

Instruction #30
Label: .L4:

Line: 60
Instruction: `MOVEQ  R0, R2`
Comments: If R0 = 0, copy the contents of R2 into R0

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988
After	*dst[2]	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1	1 or 0	1 or 0
After	1 or 0	1	1 or 0	1 or 0

Instruction #31

Label: .L4:
Line: 61
Instruction: **POP** `{R4, R5, PC}`
Comments: Balance the stack to avoid memory issues

State of Relevant Registers

	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)
Before	*dst[2]	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988
After	*dst[2]	0	*dst[2]	*dst[2]	0	0	2	0x1000

Stack Diagram

Address	Before	After
0x1000	LR	LR
0x0996	R5	R5
0x0992	R4	R4
0x0988	0	0

State of Condition Flags

	N (Negative)	Z (Zero)	V (Overflow)	C (Carry)
Before	1 or 0	1	1 or 0	1 or 0
After	1 or 0	0	1 or 0	1 or 0

Appendix A

e.g stpncpy(dst[5], src[5], 2)				Condition Flags				General Purpose	General Purpose	General Purpose	General Purpose	General Purpose	General Purpose	Intra Procedural Call	Stack Pointer	Program Counter	Stack Memory					
Label	Mnemonic	Arguments	Notes	N	Z	C	V	R0	R1	R2	R3	R4	R5	R12 (IP)	R13 (SP)	R15 (PC)	0x1000	0x1000	0x996	0x0992	0x0988	0x0980
	N/A	N/A		0	0	0	0	*dst[0]	*src[0]	2	0	x	x	N/A	0x1000		0	0	0	0	0	0
stpncpy	PUSH	{R4, R5, LR}	Store current re	0	0	0	0	*dst[0]	*src[0]	2	0	x	x	N/A	0x0988		0	LR	R5	R4	0	0
	MOV	R3, R0	Move the *dst (a	0	0	0	0	*dst[0]	*src[0]	2	*dst[0]	x	x	N/A	0x0988		0	LR	R5	R4	0	0
	MOV	IP, #0	Change the valu	0	0	0	0	*dst[0]	*src[0]	2	*dst[0]	x	x	0	0x0988		0	LR	R5	R4	0	0
	B	.L2	Branch to .L2	0	0	0	0	*dst[0]	*src[0]	2	*dst[0]	x	x	0	0x0988	.L2	0	LR	R5	R4	0	0
.L2	CMP	R2, #0	Check if count i:	0	1 or 0	0	0	*dst[0]	*src[0]	2	*dst[0]	x	x	0	0x0988		0	LR	R5	R4	0	0
	BNE	.L5	If count is not ze	0	0	0	0	*dst[0]	*src[0]	2	*dst[0]	x	x	0	0x0988	.L5	0	LR	R5	R4	0	0
.L5	LDRB	R5, [R1, IP]	Load the value :	0	0	0	0	*dst[0]	*src[0]	2	*dst[0]	x	*src[0 + 0]	0	0x0988		0	LR	R5	R4	0	0
	SUBS	R2, R2, #1	Decrement cou	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0]	*src[0]	1	*dst[0]	x	*src[0]	0	0x0988		0	LR	R5	R4	0	0
	STRB	R5, [R0, IP]	Copy the value	0	1 or 0	0	0	*dst[0] = *src[0]	*src[0]	1	*dst[0]	x	*src[0]	0	0x0988		0	LR	R5	R4	0	0
	ADDS	R4, R3, #1	Increment *dst[(	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0] = *src[0]	*src[0]	1	*dst[0]	*dst[0 + 1]	*src[0]	0	0x0988		0	LR	R5	R4	0	0
	ADD	IP, IP, #1	Increment the c	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0] = *src[0]	*src[0]	1	*dst[0]	*dst[1]	*src[0]	1	0x0988		0	LR	R5	R4	0	0
	CBNZ	R5, .L3	Detemine if the	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0] = *src[0]	*src[0]	1	*dst[0]	*dst[1]	*src[0]	1	0x0988	.L3	0	LR	R5	R4	0	0
	MOV	R0, R3	If the current ch	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0] = *src[0]	*src[0]	1	*dst[0]	*dst[1]	*src[0]	1	0x0988		0	LR	R5	R4	0	0
	MOV	R3, R4	Move the *dst (a	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0] = *src[0]	*src[0]	1	*dst[1]	*dst[1]	*src[0]	1	0x0988		0	LR	R5	R4	0	0
.L3	MOV	R3, R4	Move the *dst (a	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0] = *src[0]	*src[0]	1	*dst[1]	*dst[1]	*src[0]	1	0x0988		0	LR	R5	R4	0	0
.L2	CMP	R2, #0	Check if count i:	0	1 or 0	0	0	*dst[0] = *src[0]	*src[0]	1	*dst[1]	*dst[1]	*src[0]	1	0x0988		0	LR	R5	R4	0	0
	BNE	.L5	If count is not ze	0	0	0	0	*dst[0]	*src[0]	1	*dst[1]	*dst[1]	*src[0]	1	0x0988	.L5	0	LR	R5	R4	0	0
.L5	LDRB	R5, [R1, IP]	Load the value :	0	0	0	0	*dst[0]	*src[0]	1	*dst[1]	*dst[1]	*src[1]	1	0x0988		0	LR	R5	R4	0	0
	SUBS	R2, R2, #1	Decrement cou	1 or 0	1 or 0	1 or 0	1 or 0	*dst[0]	*src[0]	0	*dst[1]	*dst[1]	*src[1]	1	0x0988		0	LR	R5	R4	0	0
	STRB	R5, [R0, IP]	Copy the value	0	1 or 0	0	0	*dst[1] = *src[1]	*src[0]	0	*dst[1]	*dst[1]	*src[1]	1	0x0988		0	LR	R5	R4	0	0
	ADDS	R4, R3, #1	Increment *dst[1	1 or 0	1 or 0	1 or 0	1 or 0	*dst[1] = *src[1]	*src[0]	0	*dst[1]	*dst[1 + 1]	*src[1]	1	0x0988		0	LR	R5	R4	0	0
	ADD	IP, IP, #1	Increment the c	1 or 0	1 or 0	1 or 0	1 or 0	*dst[1] = *src[1]	*src[0]	0	*dst[1]	*dst[2]	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	CBNZ	R5, .L3	Detemine if the	1 or 0	1 or 0	1 or 0	1 or 0	*dst[1] = *src[1]	*src[0]	0	*dst[1]	*dst[2]	*src[1]	2	0x0988	.L3	0	LR	R5	R4	0	0
.L3	MOV	R3, R4	Move the *dst (a	1 or 0	1 or 0	1 or 0	1 or 0	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988		0	LR	R5	R4	0	0
.L2	CMP	R2, #0	Check if count i:	0	1 or 0	0	0	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	BNE	.L5	If count is not ze	0	0	0	0	*dst[1] = *src[1]	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988	.L5	0	LR	R5	R4	0	0
	MOV	R0, R2	If count is zero,	0	1	0	0	0	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	B	.L6	Branch to .L6	0	1	0	0	0	*src[0]	0	*dst[2]	*dst[2]	*src[1]	2	0x0988	.L6	0	LR	R5	R4	0	0
.L6	MOV	R4, R2	Count variable t	1 or 0	1 or 0	1 or 0	1 or 0	0	*src[0]	0	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	MOVS	R1, #0	Clear the *src a	1 or 0	1 or 0	1 or 0	1 or 0	0	0	0	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	B	.L4	Branch to .L4	1 or 0	1 or 0	1 or 0	1 or 0	0	0	0	*dst[2]	0	*src[1]	2	0x0988	.L4	0	LR	R5	R4	0	0
.L4	CMP	R4, #0	Check if the cou	1 or 0	1 or 0	1 or 0	1 or 0	0	0	0	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	BNE	.L7	If count is not e	1 or 0	0	1 or 0	1 or 0	0	0	0	*dst[2]	0	*src[1]	2	0x0988	.L7	0	LR	R5	R4	0	0
	ADDS	R2, R3, R2	Add the values	1 or 0	1	1 or 0	1 or 0	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	CMP	R0, #0	Check if the valu	1 or 0	1 or 0	1 or 0	1 or 0	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	IT	EQ	Execute IT Bloc	1 or 0	1	1 or 0	1 or 0	0	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	MOVEQ	R0, R2	If R0 = 0, copy t	1 or 0	1	1 or 0	1 or 0	*dst[2]	0	*dst[2]	*dst[2]	0	*src[1]	2	0x0988		0	LR	R5	R4	0	0
	POP	{R4, R5, PC}	Balance the sta	1 or 0	0	1 or 0	1 or 0	*dst[2]	0	*dst[2]	*dst[2]	x	x	2	0x1000	LR	0	LR	R5	R4	0	0
	EOF							ReturnValue														