Part 7: Calling Conventions

A function is a section of a program that performs a specific task. In order to call a function, and depending on the type of function, specific conditions should be met. These standards are defined as calling conventions.

As an analogy, there are different jobs in our society and a baker might get ready in the morning differently than a teacher gets ready in the morning.

Using the code example below, we will discuss the different calling conventions of:

- Stem and Leaf Function calls
- Library calls
- System calls

```
#include <stdio.h>

int multiply(int a, int b) {
    return a * b;
}

int moreThanFour(int a, int b, int c, int d, int e, int f) {
    printf("%d %d %d %d %d %d\n", a, b, c, d, e, f);
    return 0;
}

int main() {
    int x = multiply(5, 4);
    moreThanFour(1, 2, 3, 4, 5, 6);
    printf("This is a library call to printf, x = %d\n", x);
    write(1, "This is a syscall to write\n", 27);
    return 0;
}
```

Function Calls

Functions that you write in your code will abide by the regular function calling convention.

This involves passing the first four arguments through the registers $a0-$a3 and saving the return address in register $ra.

A function that calls other functions is a **stem** function whereas a function that does not call any functions is a **leaf** function.

Stem functions save $ra onto the stack to preserve it before calling another function and using the register whereas leaf functions can just use the register.

Similarly, there are several other registers that are preserved across function calls. They are the **saved** registers $s0-$s8. Across function calls, they are saved onto the stack before the next function call.

In the event that more than four arguments are passed to a function, all arguments after the fourth are stored onto the stack.

This can be seen in the MIPS assembly below

```
// Equivalent C code: moreThanFour(1, 2, 3, 4, 5, 6);

960:   24020006        li      v0,6
964:   afa20014        sw      v0,20(sp)    # argument 6 is stored on the stack
968:   24020005        li      v0,5
96c:   afa20010        sw      v0,16(sp)    # argument 5 is stored on the stack
970:   24070004        li      a3,4         # arguments 1-4 are stored in $a0-$a3
974:   24060003        li      a2,3
978:   24050002        li      a1,2
97c:   24040001        li      a0,1
980:   8f828038        lw      v0,-32712(gp)
984:   0040c825        move    t9,v0
988:   0411ffc2        bal     894 <moreThanFour>    # bal will save the address 990 in $ra
98c:   00000000        nop                           # branch delay slot
990:   8fdc0018        lw      gp,24(s8)    # after moreThanFour() finishes, code will return here
```

Library Calls

Functions from C standard libraries such as printf() are compiled to follow the convention of loading the address of the library function into register $t9 and then jumping to $t9.

This is important because $t9 will be used to offset other things in the library code.

Generally when writing C code, we don't have to worry about this as the compiler takes care of this for us; however, if you are writing MIPS assembly and want to call a library function, make sure to load the library function's address into $t9.

All of the previous aspects of a function call as described above apply - passing arguments to $a0-$a3 and saving $ra.

This can be seen in the MIPS assembly below

```
// Equivalent C code: printf("This is a library call to printf, x = %d\n", x);

994:   8fc50020        lw      a1,32(s8)       # argument 1 is x
998:   8f828030        lw      v0,-32720(gp)
99c:   24440b44        addiu   a0,v0,2884      # argument 0 is the string "This is a library call to printf, x = %d\n"
9a0:   8f828068        lw      v0,-32664(gp)
9a4:   0040c825        move    t9,v0           # load address of printf into $t9
9a8:   0320f809        jalr    t9              # jump to $t9
9ac:   00000000        nop
```

System Calls

System calls or **syscalls** are special functions used to directly communicate with the operating system. During a system call, control is passed to the operating system to perform the syscall before communicating back to the user via the return value.

Many standard library functions are wrappers for syscalls, for example printf() utilizes the write() syscall to output to standard out.

A short list of possible system call functions is:

- write()
- read()
- socket()

To perform a syscall, the syscall code (number) is specified in the register $v0 and arguments are passed in registers $a0-$a3.

Then the syscall instruction is performed.

```
li $v0, syscall_number
syscall 0x404040
```

Syscall codes can be found in `/usr/include/mips-linux-gnu/asm/unistd.h` or at w3challs

Further Reading

1. MIPS Assembly Wikibook Subroutines

2. Phrack - Writing MIPS Shellcode

Time to practice! MIPS Lab Setup