Introduction

This tutorial is more of a tip than a tutorial. It just explains how to calculate offsets for jumps and calls within the program you are patching.

Types of Jumps/Calls

Here I will just describe the different types of jumps and calls which you will come across:

Short Jumps
Short jumps be they conditional or unconditional jumps are 2 bytes long (or 1 nibble if your Californian ;-). These are relative jumps taken from the first byte after the two bytes of the jump. Using short jumps you can jump a maximum of 127 bytes forward and 128 bytes backwards.

Long Jumps
Long jumps if they are relative are 6 bytes long for conditional jumps and are 5 bytes long for unconditional jumps. For conditional jumps 2 bytes are used to identify that it is a long jump and what type of jump (je, jg, jns etc) it is. The other 4 bytes are used to show how far away the target location is relative to the first byte after the jump. In an unconditional jump only 1 byte is used to identify it as a long unconditional jump and the other 4 are used to show it's target's relative position, as with the conditional jumps.

Calls
There are two different types of calls which we will use. The normal type of call works the same as the long jumps in that it is relative to it's current position. The other type gives a reference to a memory location, register or stack position which holds the memory location it will call. The position held by the later is direct e.g. the memory location referenced may contain 401036h which would be the exact position that you would call, not relative to the position of the call. The size of these types of calls depends on any calculations involved in the call i.e. you could do: 'call dword ptr [eax * edx + 2]'. Long jumps can also be made using this method, but I didn't say that earlier as to avoid repetition.

Tables
Here is a brief list of all the different types of jumps/calls and their appropriate op-codes. Where different jumps have the same Op-Codes I have grouped them:

Jump	Description	Short Op-Code	Long Op-Code
call	procedure call	E8xxxxxxxx	N/A
jmp	u nconditional jump	EBxx	E9xxxxxxxx
ja/jnbe	jump if above	77xx	0F87xxxxxxxx
jae/jnb/jnc	jump if above or equal	73xx	0F83xxxxxxxx
jb/jc/jnae	jump if below	72xx	0F82xxxxxxxx
jbe/jna	jump if below or equal	76xx	0F86xxxxxxxx
jcxz/jecxz	jump if cx/ecx equals zero	E3xx	N/A
je/jz	jump if equal/zero	74xx	0F84xxxxxxxx
jne/jnz	jump if not equal/zero	75xx	0F85xxxxxxxx
jg/jnle	jump if greater	7Fxx	0F8Fxxxxxxxx
jge/jnl	jump if greater or equal	7Dxx	0F8Dxxxxxxxx
jl/jnge	jump if less	7Cxx	0F8Cxxxxxxxx
jle/jng	jump if less or equal	7Exx	0F8Exxxxxxxx

jno jump if not overflow 71xx 0F81xxxxxxxx
jnp/jpo jump if no parity/parity odd 7Bxx 0F8Bxxxxxxxx
jns jump if not signed 79xx 0F89xxxxxxxx
jo jump if overflow 70xx 0F80xxxxxxxx
jp/jpe jump if parity/parity even 7Axx 0F8Axxxxxxxx
js jump if sign 78xx 0F88xxxxxxxx

Calculating Offsets (finding in the xx's in table)

You will need to be able to calculate offsets when you add jumps and make calls within and to the code you have added. If you choose to do this by hand instead of using a tool then here are the basics:

For jumps and calls further on in memory from your current position you take the address where you want to jump/call and subtract from it the memory location of the next instruction after your call/jump i.e.:

(target mem address) - (mem location of next instruction after call/jump)

Example
If we wanted to jump to 4020d0 and the next instruction *after* the jump is at location 401093 then we would use the following calculation:

4020d0 - 401093 = 103d

We then write the jump instruction in hex as e93d100000 where e9 is the hex op-code for a long relative jump and 3d100000 is the result of our calculation expanded to dword size and reversed.

For jumps and calls to locations *before* the current location in memory you take the address you want to call/jump to and subtract it from the memory location of the next instruction after your call/jump, then subtract 1 and finally perform a logical NOT on the result i.e.

NOT(mem address of next instruction - target mem address - 1)

Example
If we wanted to call location 401184 and the address of the next instruction after the call is 402190 then we do the following calculation:

NOT(402190 - 401184 - 1) = ffffeff4

We can then write our call instruction in hex as e8f4efffff where e8 is the hex op-code for relative call and f4efffff is the result of the calculation in reverse order.

If you want to practice with different examples then the best way to do this is to use a disassembler like WDASM which shows you the op-codes and try and work out the results yourself. Also as an end note you don't have to perform these calculations if you have enough room to make your jump or call instruction into an absolute jump call by doing the following as represented in assembler:

```
mov eax, 4020d0
call eax (or jmp eax)
```

Final Notes

Make life easier and use a program to do this ;-)