Optimization of `my_strchr`

Original

```
.global my_strchr

my_strchr:
        src .req r0
        c   .req r1
        i   .req r2
        t   .req r3
        ret .req r0

        mov i, #0
        loop:
                ldrb t, [src, +i]
                cmp t, #0
                beq loop_end
                cmp t, c
                beq found
                add i, i, #1
                b loop
        found:
                mov ret, i
                b end
        loop_end:
                mov ret, #-1
                b end
        end:
                bx lr
```

Optimized

```
.global my_strchr_opt

my_strchr_opt:
        src .req r0
        c   .req r1
        i   .req r2
        t   .req r3
        ret .req r0

        mov i, #0
        ldrb t, [src, +i]
        loop:
                cmp t, #0
                beq loop_end
                cmp t, c
                beq found
                add i, i, #1
                ldrb t, [src, +i]
                b loop
        found:
                mov ret, i
                bx lr
        loop_end:
                mov ret, #-1
                bx lr
```

The existing code has a number of inefficiencies that need to be fixed. First, there is a two cycle stall after the `ldrb` instruction because of the dependency on `t`. Second, there are unnecessary `b end` instructions involved in the end of the routine that can be removed.

To fix the stalls, pre-loading of t is used. This means that there will be a 2 cycle stall for the first iteration of the loop, but future iterations will not have a stall. For most loop runs, this will result in a 2 cycle savings. The original loop requires 11 cycles in the normal case, and the optimized code requires only 9 cycles.

To fix the unnecessary `b end` instructions, the end label is removed and `bx lr` is repeated (called inlining) in both places where `b end` was previously. This will save 3 cycles as the subroutine finishes, and also removes one instruction from the code.

I decided to apply these changes because they are most straightforward approach to optimizing this code. I also investigated the possibility of applying partial loop unrolling (for example, checking 4 characters per loop iteration) but if you do that then you need to add extra code to the loop to properly calculate i, which removes the efficiency gain.