

# Re: C or Assembly

---

*Source:* <http://sci.tech-archive.net/Archive/sci.electronics.basics/2006-03/msg00193.html>

---

- *From:* Andrew Merton <nobody@xxxxxxxxxxxx>
  - *Date:* Mon, 06 Mar 2006 01:15:53 +1300
- 

John Larkin wrote:

A program is easy to maintain if it's logical, has sensible data structures, and most importantly if it's thoroughly and correctly commented.

Hear, hear!

> Most C programmers comment little if any, apparently by

tradition and training. My assembly programs are extensively commented – roughly 4x as many characters of comment compared to code – and are easy to understand and maintain years later.

If variables, functions etc are named properly, then code in high-level languages is much more self documenting (than if it's not – this is not necessarily in comparison to assembler), and the need for comments is reduced. By no means removed.

And I always have a single batch file, GO.BAT, that completely rebuilds the entire project and makes a new rom file, and we archive all the tools.

Ahah – a Makefile! B-)

I think the one-operation-per-line and resulting ease of commenting make assembly a lot easier to read than the monkeys-pounding-typewriters look of C. If you read my comments, you'll know exactly what's happening and why.

But you need so many lines! And the comments have to be kept up-to-date and accurate (or they're worse

Re: C or Assembly

than not having any at all), which adds to the maintenance burden which is already exacerbated by having 10 times as many lines of code in the first place...

```
.SBTTL . XMAC : MACRO EXECUTION ENGINE

; THIS IS ENTERED WITH A0 AIMING AT A MACRO CODE FOLLOWED BY A
; PARAMETER LIST. THIS IS USUALLY IN VME SPACE, BUT CAN BE IN ROM
; WHEN THAT'S USEFUL. WE ALWAYS COPY THE COMMAND AND
PARAMETERS
; INTO THE 'MAC' SPACE IN CPU RAM, TO ALLOW ROM INVOCATIONS AND
; TO SPEED UP MACRO EXECUTIONS, ESPECIALLY FOURIER FUNCTIONS.
;
; WE'LL TRY TO EXECUTE THE MACRO, AND WILL RETURN STATUS
; IN D7. B15 WILL BE UP IF WE DETECT AN ERROR.

XMAC: CLR.W AFLAG.W ; NUKE 'ADD WAVE' REQUEST

MOVE.W # 0FF00h, D7 ; ERROR CODE 255 WILL BE 'BAD MACRO'

MOVE.W (A0), D0 ; NAB CALLER'S COMMAND CODE
ANDI.W # 255, D0 ; ISOLATE LOW BYTE
CMPI.W # MAXIM, D0 ; CHECK AGAINST LIMIT
BHI BADMAC ; POST ERROR IF OUT OF RANGE.

; NOW COPY THE ENTIRE COMMAND BLOCK...

MOVEA.W # MAC, A1 ; AIM AT DESTINATION IN CPU RAM
MOVE.W # 110, D6 ; MAKE DBF COUNT FOR COMMAND+110 PARAMS

FILCH: MOVE.W (A0)+, (A1)+ ; COPY, WORDWISE.
DBF D6, FILCH

ASL.W # 1, D0 ; SCALE CMD * 2 FOR WORD LOOKUP
ADDI.W # DISMAL, D0 ; ADD IN TABLE START
MOVEA.W D0, A1 ; COPY TO AN ADDRESS REG

MOVEA.W (A1), A1 ; POP THE ROUTINE ADDRESS
JMP (A1) ; AND HOP TO IT.

; ALL COMMAND ERRORS JUMP HERE:

BADMAC: ORI.W # B15, D7 ; POST THE 'MACRO ERROR' CODE
CLR.B D7 ; AND NUKE 'COMMAND CODE' FIELD
RTS ; (BITS 8-14 HOLD ERROR CODES)

; AND SUCESSFUL MACROS EXIT HERE:

OKAY: CLR.W D7 ; HERE, INDICATE GOODNESS
```

RTS

.SBTTL . MACRO COMMAND DISPATCH TABLE

DISMAL: .WORD BADMAC ; 00 ILLEGAL COMMAND  
.WORD FOURS ; 01 BUILD FOURIER SERIES

> <<SNIP>>

.WORD URAM ; 26 TEST CPU SRAM

MAXIM = 26h ; LAST LEGAL MACRO CODE

I think this would be a fair replacement for the above code (but my C and Assembler are both a bit rusty, and this is completely uncompiled and untested so I'm probably wrong...):

```
#define MAXIM 0x26
```

```
int BADMAC(int *args) {  
    return 0x80ff;  
}
```

```
/* other function definitions/prototypes... */
```

```
int *DISMAL[] = {  
    BADMAC,  
    FOURS,  
    ...,  
    URAM  
};
```

```
/* Given an index into the DISMAL table, and a 110–element array  
of integer arguments, execute the specified function.
```

```
All the called functions must have the same signature as  
BADMAC, above. */
```

```
int XMAC(int cmd, int *args) {
```

```
    /* Check the parameter */  
    if (cmd > MAXIM) {  
        return 0x80ff;  
    }
```

```
    /* Index the array and call the function */  
    return (DISMAL[cmd])(args);
```

```
}
```

Re: C or Assembly

IMO, the C is easier, shorter and more succinct. Being so much shorter, it also will lead to fewer defects and therefore shorter development and test times.

Note that I'm not saying this is the best way to do this in C...

I guess my real point is that since C programs are shorter, they need less comments, but still need "enough".

JM2c...

.