then this stage will probably be the easiest and least time-consuming of all. In order to translate from a high-level algorithm to low-level code it is essential that the control structures used at high level are carried over to the low level, avoiding the temptation to use BRA and JMP indiscriminately. Remember that any time you save by writing unstructured code is certain to be 'clawed back' in a frustrating trial-and-error debugging stage. In the diagram we give some examples of the way in which the common control structures can be coded — assuming, for simplicity, that the data items used are eight-bit.

One problem with coding with control structures in this way is that the program is longer than it might be. Where space is not limited then there is no point in trying to save it; short code does not usually mean shorter running times but it does mean longer development and debugging times. Where space is limited, then it is better to write in a spacious structured way, and introduce a further stage of optimisation where the working code can be shortened to take into account particular circumstances, retaining as far as possible the essential structure.

- Debugging: At this stage, each module is separately tested — using stubs where necessary — to make sure it gives appropriate outputs for valid inputs. Debugging Assembly language programs differs considerably from BASIC program debugging. To be able to see what is happening, it is necessary to be able to inspect the contents of the registers and the memory locations used by the program, and to change them if necessary. It is nearly impossible to debug an Assembly program without the use of a utility for setting and removing breakpoints. These enable you to run the program up to the next breakpoint, then dump the registers, and inspect and change memory contents.
- Testing: Once each module has been tested and debugged then the entire program has to be put together and tested with appropriate data. This is much easier when you know that all the component parts are working properly.
- Documentation: Assembly language programs are more difficult to understand than high-level programs, so documentation is even more important. In particular, it is vital to document the use of memory, the use of the stack (especially while passing parameters), and the register usage within subroutines.
- Maintenance: If a program is to be used over a period of time then at some point it will probably need revision — either to remove any bugs that appear or to make improvements. It is at this stage that time spent in careful design and documentation really pays off. If the program is badly designed and/or poorly documented then you are better off doing a complete rewrite rather than attempting to make alterations.

Now we need a project to apply these design skills to: for our first venture in structured Assembly language programming nothing could be more appropriate than a machine code monitor/debugger. If you've used an assembler before, then you may be familiar with the kind of utilities to expect from a monitor/debugger. Essentially, it gives the machine code programmer the kind of editing facilities that the BASIC programmer takes for granted — namely, the ability to inspect and change the contents of memory.

In the next instalment of the course we will take this project through the design and development stages described in this article, to create an important and extremely useful programming aid.

Basic Backbone
There are no control structures written into Assembly language, so it pays to import tried and tested methods from high-level languages. The structures shown here are clear and graceful in both high- and low-level languages, and should be used to the exclusion of all alternatives

Control Structures

The IF...THEN...ELSE Structure

Pseudo-code	Assembly Language
	THREE FCB 3
IF NUM1 = 3	IF LDA NUM1
THEN	CMPA THREE
routine1	BNE ELSE
	THEN .....
ELSE	* routine1
routine2	BRA ENDIF
ENDIF	ELSE .....
	* routine 2
	ENDIF .....

The WHILE...WEND Structure

Pseudo-code	Assembly Language
WHILE NUM1 <= 3	WHILE LDA NUM1
repeated routine	CMPA THREE
WEND	BGT WEND
	* repeated routine
	BRA WHILE
	WEND .....

The REPEAT...UNTIL Structure

Pseudo-code	Assembly Language
REPEAT	REPEAT .....
repeated routine	* repeated routine
UNTIL NUM1 < 3	LDA NUM1
	CMPA THREE
	BGE REPEAT
	UNTIL .....

The FOR...NEXT Structure

Pseudo-code	Assembly Language
FOR NUM1 = 1 TO NUM2	LDA NUM2
repeated routine	FOR .....
NEXT NUM1	*repeated routine
	DECA
	BGT FOR
	NEXT .....