INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
34100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONES: 224281/2/3/4/5/6
CABLE: CENTRATOM - TELEX 460392-1

SMR/90 - 29

COLLEGE ON MICROPROCESSORS:

TECHNOLOGY AND APPLICATIONS IN PHYSICS

7 September - 2 October 1981

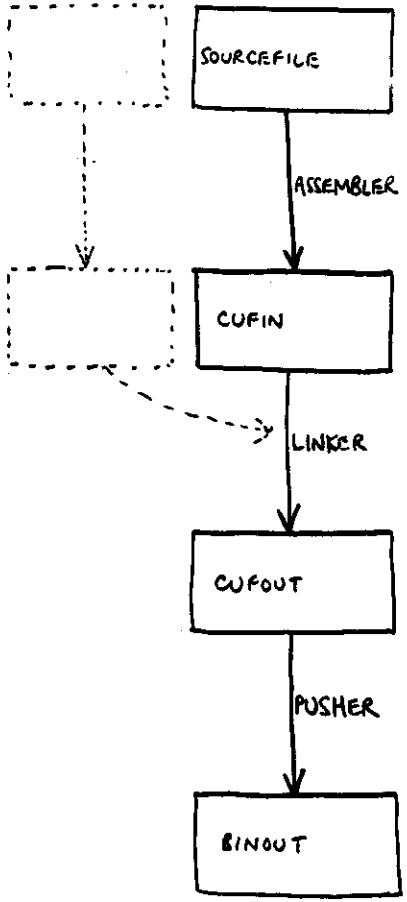
ADVANCED FEATURES OF THE ASSEMBLER

M. BARTON

Dept. of Computation
Institute of Science and Technology
University of Manchester
P.O. Box 88
Manchester M60 1QD
England

INTRODUCTION TO EXERCISE 6·1 : EXERCISE ON ASSEMBLER & LINKER

A : BACKGROUND



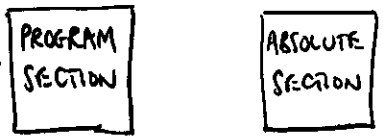
- source program (assembly language)
- relocatable object modules with unresolved external references
- relocatable object module with NO unresolved external references
- absolute load module

RELOCATION ("PUSHING")

- Programmer can write program without knowing the actual addresses of the memory in the system.

```
                    OPT   REL   <- puts assembler into 'relocatable' mode
                    PSCT        <- introduces a program section
0000
                    DSCT        <- introduces a data section
0000
                    ASCT        <- introduces an absolute section
                    ORG $4000
4000
```

OUTPUT OF ASSEMBLER WOULD BE :-



AT PUSH TIME, USER MUST SPECIFY ABSOLUTE ADDRESS FOR THESE 2 SECTIONS. NORMALLY WOULD PUT PROG. SECTION INTO ROM AND DATA SECTION INTO RWM.

"relocatable object module"

LINKING

- Can assemble and test individual routines separately.
- Then build system from object modules — no need to reassemble.

PROBLEM:

When writing program, will need to refer to addresses of data/subroutines in other modules : i.e. EXTERNAL REFERENCES.

e.g.

MODULE 1

```
:
JSR  FRED   } call of "FRED"
:
```

MODULE 2

```
:
FRED  —————  } subroutine "FRED"
:
```

∴ IN MODULE 1, must write XREF FRED } at top of program

IN MODULE 2, must write XDEF FRED

SUCCESSFUL LINKING requires that all external references be resolved;

i.e. every XREF must have a matching XDEF.

LIBRARIES

To build a system from relocatable object modules, the user must give the LINKER the names of the modules required:

MODULE1, MODULE2, ...

The linker will produce a system containing all these modules.

IT IS CONVENIENT to collect modules which provide service routines (e.g. input/output, multiply/divide) into a LIBRARY. So instead of writing

MODULE1, MODULE2, ARITHA, ARITHB, ARITHC, INOUT . . .

the user need only write

MODULE1, MODULE2, LIBRARY.

The only modules from the LIBRARY which will be used are those needed to satisfy the external references in MODULE1 & MODULE2.

B : SPECIFICATION OF THE PROBLEM

- Write, assemble, & test a module which provides a subroutine "DSPLAY".
  DSPLAY is to produce a histogram on the Newbury VDU, showing the information stored in memory at address HIST.

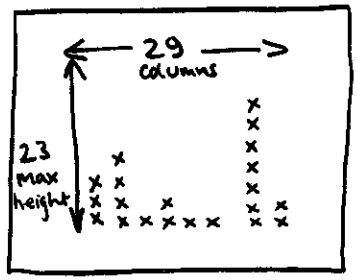
MODULE to be written.

```
XREF   HIST
XDEF   DSPLAY

DSPLAY  ---
        ---
        ---
        ---
        ---
        RTS
```

MAIN MODULE

```
XREF DSPLAY
XDEF HIST

     ---
     ---
     JSR DSPLAY
     ---
HIST ---  3
     ---  2
     ---  1
```

N.B. We also need to write a main module to test our DSPLAY module.



VDU-showing histogram
[Histogram must be centred.]

C : SUGGESTED APPROACH

Histogram is built up row-by-row, starting at the top L.H. corner.

Algorithm (in mixture of high-level language + English) :

```
WRITE Line feed, Carriage return.
FOR LINE = 23 TO 1 STEP -1
|
|   WRITE 25 SPACES FOR MARGIN
|
|   RESET HIST POINTER TO START OF HIST TABLE
|
|   FOR COLUMN = 1 TO 29
|   |
|   |   IF LINE > HIST TABLE ENTRY THEN WRITE SPACE
|   |                              ELSE WRITE 'X'
|   |
|   |   STEP HIST POINTER
|   |
|   NEXT COLUMN
|   WRITE Line feed, Carriage return.
|
NEXT LINE
```

SOPHISTICATED FEATURES OF ASSEMBLER

- **MACROS**

Facilitate the writing of sections of assembly language program which are repeated.

e.g. if we define a macro "BILL":

```
BILL    MACR
        ====   } "X"
        ENDM
```

then subsequently, every appearance of "BILL" will be translated into "X" — saves writing "X" out in full each time.

"X" need not be exactly the same every time: Macros can take arguments.

e.g.

```
BILL    MACR        <- "MACRO DEFINITION"
        ----
        \0
        ----
        \1
        ----
        \0
        ----
        ENDM
```

/over

The assembler will expect appearances of "BILL" to be followed by 2 arguments.

e.g.

```
BILL    JOHN, JIM   <- "MACRO CALL"
```

would be translated as

```
        ----
        ----
        JOHN
        ----
        JIM
        ----
        ----
        JOHN
        ----
```

- **EXERCISE ON MACROS**

In problem 6.1, use a macro "MARGIN" to write the margin spaces at the start of each line. The macro should take one argument, indicating the number of spaces to be written.

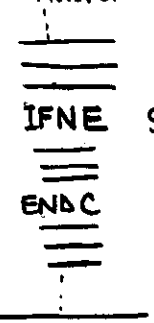
So the macro call would look like:

```
MARGIN  25
```

• CONDITIONAL ASSEMBLY

Facilitates the assembly of various versions of an assembly language program — all based upon a standard master program.

e.g. Master program might be:



```
        IFNE  SPCIAL
        ...
        ENDC
```

} we want this portion to be included only in the "special" version of the program.

AT TOP OF PROGRAM, must put either

```
SPCIAL  EQU  0   (for 'ordinary' version)
or
SPCIAL  EQU  1   (for 'special' version)
```

• EXERCISE ON CONDITIONAL ASSEMBLY

In problem 6.1, use a symbol "MFLAG" to control the production of the margin. If MFLAG=0, the code which writes the margin spaces is to be suppressed.

D : SUMMARY OF WORK FOR EX. 6.1

- Write + assemble* module containing DSPLAY
- If assembly errors, correct source + reassemble
- Write + assemble "main module" (containing test histogram data).
- If assembly errors, correct source + reassemble.
- Link and push the two modules.
- If link or push errors, correct source + reassemble + relink/push.
- Run the system.
- If execution errors, correct source etc...

*If time available, try producing different versions using conditional assembly.

M.H. Barton
15SEP81