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-I

SMR/101 - 17

SECOND COLLEGE ON MICROPROCESSORS: TECHNOLOGY AND APPLICATIONS IN PHYSICS

(18 April - 13 May 1983)

VARIOUS TRANSPARENCIES

C. HALATSIS

Digital Systems and Computer
Laboratory
Aristotle University of Thessaloniki
Thessaloniki
Greece

SOFTWARE TOOLS

TAXONOMY OF TOOLS

	SOFTWARE LIFE CYCLE		
	CONCEPTUAL & REQUIREMENTS	DEVELOPMENT	OPERATIONS & MAINTENANCE
CLASSIFICATION SCHEME			
SIMULATION	x	x	x
DEVELOPMENT	x	x	
TEST & EVALUATION		x	
OPERATIONS & MAINTENANCE			x
PERFORMANCE MEASUREMENT		x	x
PROGRAMMING SUPPORT	x	x	x

1

CONVENTIONAL SOFTWARE TOOLS FOR SINGLE-CHIP FIXED INSTRUCTION SET μP's.

- TEXT EDITORS
- LANGUAGE TRANSLATORS
  - ASSEMBLERS
  - COMPILERS
  - INTERPRETERS
- LOADERS & LINKERS
- SIMULATORS
- DEBUGGERS

- SYSTEM SOFTWARE
  - OPERATING SYSTEM
  - FILE MANAGEMENT
  - UTILITIES

CROSS-COMPUTER TOOLS:

THESE ARE TOOLS WHICH RUN ON SOME HOST COMPUTER OTHER THAN THE μP FOR WHICH SOFTWARE IS BEING DEVELOPED

RESIDENT TOOLS:

THEY RUN ON THE SAME μP FOR WHICH SOFTWARE IS BEING DEVELOPED

TRADEOFFS BETWEEN USING CROSS-TOOLS VERSUS RESIDENT

CROSS COMPUTERS ARE GENERALLY FASTER

LESS INITIAL COSTS FOR CROSS TOOLS

OVERALL COSTS HIGHER FOR CROSS TOOLS

CROSS TOOLS MORE CONVENIENT TO USE

CROSS TOOLS GENERALLY MORE POWERFUL

CROSS TOOLS LACK REAL-TIME DEBUGGING

TEXT EDITORS

- They allow you to enter, modify, and store <u>text</u> in a file

{INSERT, DELETE, REPLACE, COPY} × {SINGLE CHARACTER, STRING OF CHARACTERS, SINGLE LINE, RANGE OF LINES, SINGLE PAGE, RANGE OF PAGES, SINGLE FILE, SET OF FILES}

- General purpose text editors
  "CONTEXT FREE"

{MOVE, FIND} EDITING WINDOW
EDITOR POINTER

- Special purpose text editors
  "CONTENT SENSITIVE"
  Example: a text editor for developing FORTRAN programs. In this case this FORTRAN-editor performs also syntax checking.

EDITING LEVELS — COMMAND, TEXT

Text editor typical commands

- LINE Editors
- PAGE Editors /SCREEN/CRT

COMMANDS OF THE XEDIT Editor

Call it : XEDIT
XEDIT, filename
XEDIT, filename,P

DEFTAB ;	define ";" as tab character
TABS, 9, 16, 32	Select tab settings
TOP	Move to beginning of edit buffer
BOTTOM	Move to end of edit buffer
LOCATE 'string'	Locate string 'string'
NEXT	Move to next line
NEXT -1	Move to previous line
DELETE	Delete current line
MODIFY	Modify current line
CHANGE /string1/string2/ n	n times
INPUT .	Insert lines of text after current line
INSERT	Insert a line after current line
INSERTB	Insert a line before current line
PRINT	print current line
PRINT n	print n lines from current line
↑P *	print all lines from beginning of file
FILE fn	Copy edit file to local file fn
STOP	Quit
END, fn	End, save edit file in fn
END, fn, S	End, save edit file in permanent fn
END, fn, R	End, replace fn with edit file 7

Commands of the DAN Editor

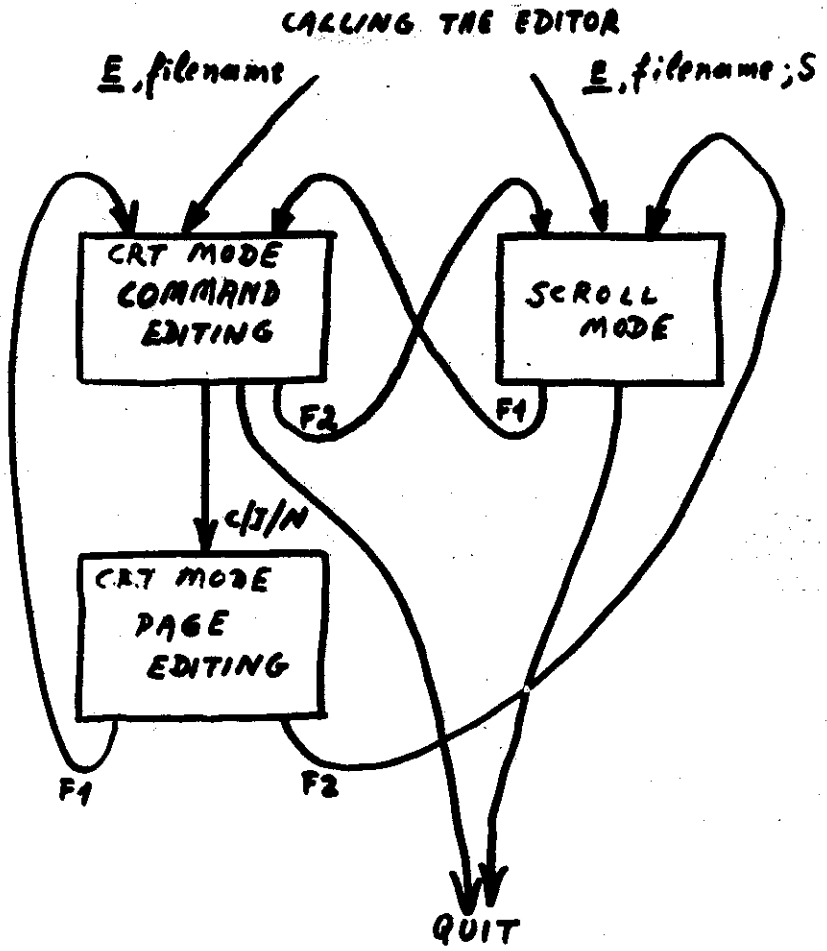
Call it DAN, lfn

ADD [,{line, CURRENT}[,increment]][,OVERWRITE
BYE
* CHANGE, /text1/, /text2/[, linerange][,VETO][,NEXT]
CREATE [,line[,increment]]
* DELETE, linerange[,VETO][,text selection][,NEXT]
EDIT, lfn[,text selection][,TRUNCATE][,SEQUENCE][,DISPLAY]
FORMAT [,SHOW][,compiler name][,CH=linelength]
INIT
INSERT, lfn[{line, CURRENT}[,increment]][,NONCHECK]
* LIST [,linerange][,textselection][,SUP][,NEXT
RESEQ[,line[,increment]]
ROUTE, DC=dispcode[,TID=tid][,FID=fid][,ST=stid]
RUN, compiler name[,NOEX][,SUP]
* SAVE, lfn[,linerange][,text selection][,VETO][,MERGE]
[,OVERWRITE][,NOSEQ][,DISPLAY]

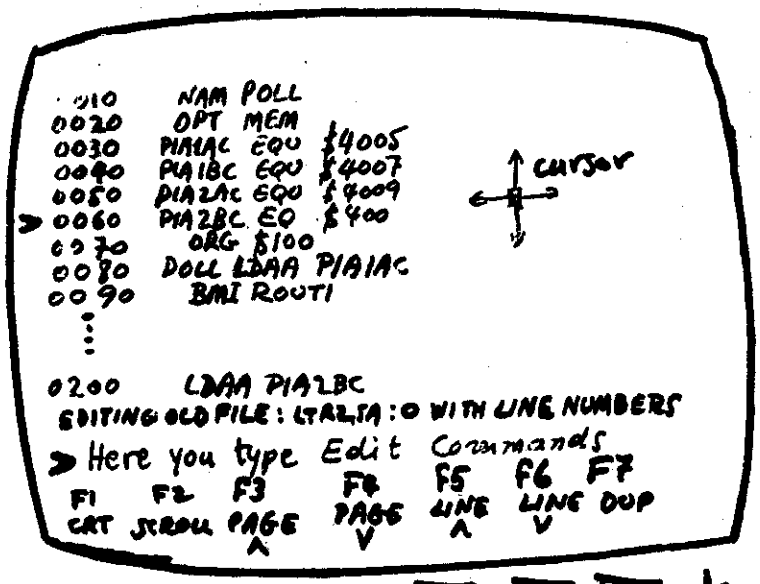
/text1/=/text2/[,line range][,VETO][,NEXT]
[,text condition][,column range][,UNIT]

text selection =/text/[,column range][,text condition][,UNIT]

A SIMPLE SCREEN/PAGE EDITOR



- In command mode the user edits his file by typing appropriate commands
- In page mode the user edits his file by moving the cursor around & changing characters by overtyping



Standard keyboard

TYPICAL COMMANDS

COMMAND	PAGE
C change	← →
F Find	↑
I Insert	↓
L List	control-X Cancel this line
N Number	TABS
S Search	DEL
SAVE	BREAK
1-9999	INS CHAR
	DEL CHAR
	LINE ERASE

Keys used when in Page Mode

* DELETE / PICK : PUT { character / word / paragraph }

ASSEMBLY LANGUAGE

- SYMBOLIC FORM OF THE MACHINE LANGUAGE
  - MNEMONICS FOR OP CODES
  - SYMBOLIC NAMES FOR
    - INSTRUCTION ADDRESSES (LABELS)
    - OPERANDS
    - OPERAND ADDRESSES
    - LITERALS
- ONE-TO-ONE CORRESPONDENCE BETWEEN

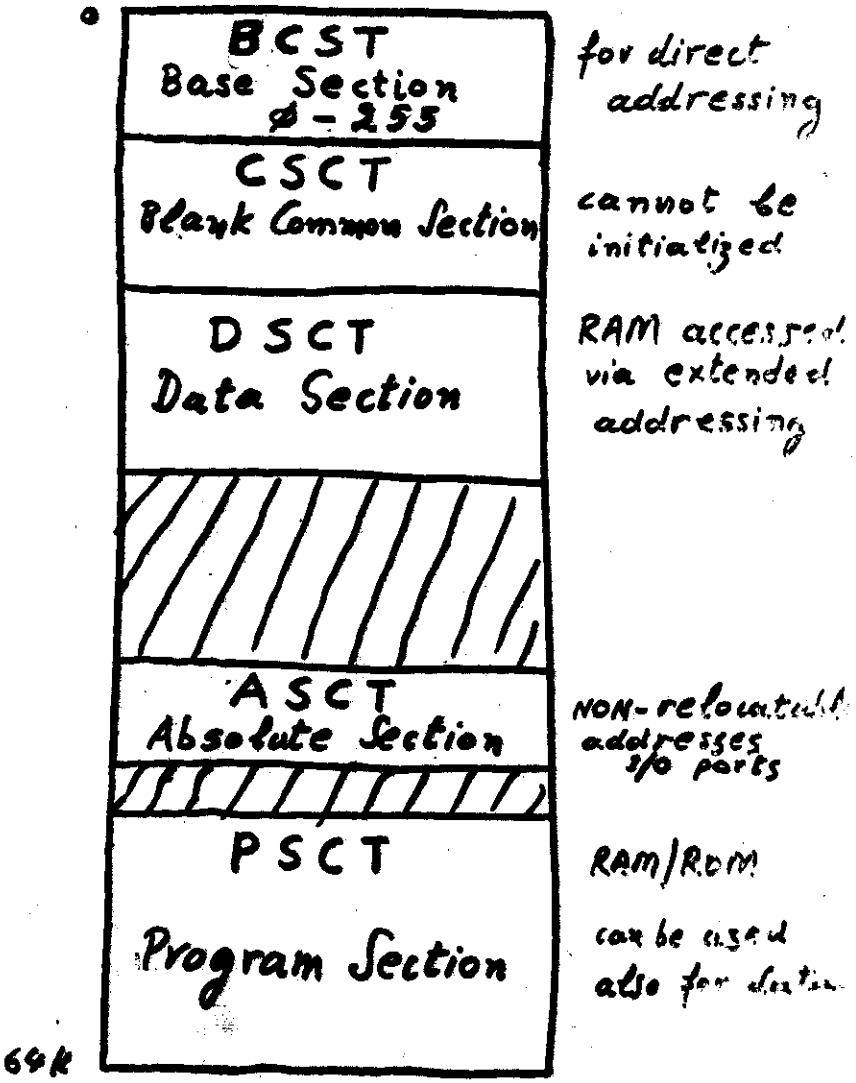
  ASSEMBLY INSTRUCTONS $\Leftrightarrow$ MACHINE INSTRUCTIONS
- PSEUDO INSTRUCTIONS $\Rightarrow$

  DIRECTIVES TO THE ASSEMBLER
- FORMAT : line number/Label/Operation/Operands/Comments

- Source, object and Listing Formats which are Easy to Use, Read, and Understand
- Ability to Define and Manipulate Meaningful Symbols
- Ability to Specify Data Constants in a Meaningful Form
- Ability to Specify an Arithmetic or Logical Expression, Evaluate it, and Use it as an operand in an instruction or directive.
- Provision of alphabetical listing of symbols with their values.
- Provision of alphabetically sorted cross-Reference listing of all symbols along with the statement defining the symbol and the statements referring to this symbol.
- Good Error Diagnostics
- Parameterized Macro Facility
- Conditional Assembly Facility
- Relocatable Object Code
- Provision of Linkage Editing Capability

Features of an assembler to look for

PROGRAM SECTIONS OF M ASSEMBLY



M ASSEMBLY LANGUAGE CONSTRUCTS

- CHARACTER SET
  - upper case letters A to Z
  - digits 0-9
  - arithmetic operators + - * /
  - two-char operators !A !O !X !+ !R !L
  - parentheses in expressions ( )

- PREFIXES FOR CONSTANTS & ADDRESSING MODES

#	immediate addressing		LDAA #$F1
$	Hex	constant	$A2
&	Decimal		&10
@	Octal		@123
%	Binary		%010
'	ASCII char		'A

- SUFFIXES FOR CONSTANS & ADDRESSING MODES

,X	indexed addressing	LDAA 2,X
H	Hex	A2H
O	Octal	123O
Q	Octal	123Q
B	Binary	11011010B

- Special symbols
  - * comment (if in column 1)
  - \ , \. macro parameters, labels

```
BRA   *+10    Instruction Location Counter
LDX   # *
```

- SYMBOLIC NAMES 1 to 6 alphanumeric char
- EXPRESSIONS combination of symbols, constants, algebraic ops and parentheses

M PSEUDO-INSTRUCTION

ASSEMBLY CONTROL

NAM, OPT {REL / OBJ / LOAD / MEM}, ORG, {ASCT / BSCT / CSCT / DSCT / PSCT}, COMM, FAIL, END

LINKING LOADER CONTROL

IDNT, XDEF, XREF

SYMBOL DEFINITION

EQU, SET, MACR, ENDM

DATA DEFINITION/STORAGE ALLOCATION

FCC, FCB, FDB, FSZ, RMB

CONDITIONAL ASSEMBLY

IF {GT / GE / LT / LE / EQ / NE}, IFC, IFNC, ENDC

LISTING CONTROL

PAGE, TTL, OPT {MEX MC / SYM MD / CLIST PAGE / CREF LIST}

MACROS

- The macro facility allows the programmer to define additional instruction mnemonics which can then be used repeatedly

- Macro definition: it specifies
  - The name of the macro instruction
  - Possibly a list of formal parameters
  - The meaning of the macro instruction (the macro body) which is normally a piece of assembly code

- Macro call: it is the use of a macro name as an instruction

- Macro expansion: it is the replacement of a macro name by its body during the assembly process (not during execution)

- Generally, a MACRO is an abbreviation for a piece of text (the body)

MACROS without parameters

Example

```
* macro definition of the macro SWAP
* it swaps accumulators  A ⇄ B
*
SWAP  MACR
      PSH A        stack ← A
      TBA             A ← B
      PUL B           B ← stack
      ENDM

* macro call
      SWAP

* macro expansion
      SWAP
36    PSH A
17    TBA
33    PUL B
```


MACROS with parameters

Example

```
* macro  EXCHNG X,Y exchanges the
* contents of memory locations X ⇄ Y

EXCHNG  MACR
        PSH A              save A
        LDA A  \0
        PSH A
        LDA A  \1
        STA A  \0
        PUL A
        STA A  \1
        PUL A              restore A
        ENDM

* macro call
        EXCHNG Z,Y
* macro expansion
        PSH A
        LDA A  Z
        PSH A
        LDA A  Y
        STA A  Z
        PUL A
        STA A  Y
        PUL A

* macro expansion
        EXCHNG  W,T
        PSH A
        LDA A  W
        PSH A
        LDA A  T
        STA A  W
        PUL A
        STA A  T
        PUL A
```

ΦΕΛΙΑΙΜΕΝΕ MACRO ΚΑΜΕΙΣ

NESTED MACRO CALLS

when a macro call is contained into the body of another macro

:xample

```
* this macro performs a left-circular
* permutation of its arguments A1,A2,A3
CYCLE   MACR
        EXCHNG \0,\1
        EXCHNG \1,\2
        ENDM
* macro expansion
        CYCLE X1,X2,X3
        EXCHNG X1,X2
        PSHA
        LDAA X1
        PSHA
        LDAA X2
        STAA X1
        PULA
        STAA X2
        PULA
        EXCHNG X2,X3
        PSHA
        LDAA X2
        PSHA
        LDAA X3
        STAA X2
        PULA
        STAA X3
        PULA
```

- NESTED MACRO DEFINITIONS

WHEN A MACRO DEFINITION IS CONTAINED INTO THE DEFINITION OF ANOTHER MACRO

ASSEMBLER GENERATED LABELS IN MACROS

Example :

Definition :

```
CALL  MACR      ; Call routine
      JSR \0
      BCC \.0   ; Skip if o.k
      JMP ERROR ; else go to ERROR
\.0   EQU *
      ENDM
```

```
INPUT  EQU $2000
OUTPUT EQU $3000
ERROR  EQU $4000
```

Call Expansion :

```
       CALL INPUT
       JSR  INPUT
       BCC  .00000
       JMP  ERROR
.00000 EQU  *
```

Call Expansion

```
       CALL OUTPUT
       JSR  OUTPUT
       BCC  .00001
       JMP  ERROR
.00001 EQU  *
```

CONDITIONAL ASSEMBLY
CONDITIONAL MACRO EXPANSION

Example

```
* macro EXCHNG Y,Z exchanges Y⇄Z, where
* X,Z may be either memory locations OR
* the accumulator A
* the macro call EXCHNG W,W does not
* produce any expansion

EXCHNG  MACR
        IFNC  \0,\1
          IFC   \0,A        case EXCHNG A,M
            PSH B
            LDA B \1
            STA A \1
            TBA
            PUL B
          ENDC
          IFC   \1,A        case EXCHNG M,A
            PSH B
            LDA B \0
            STA A \0
            TBA
            PUL B
          ENDC
          IFNC  \0,A
            IFNC  \1,A      case EXCHNG M1,M2
              PSH A
              LDA A \0
              PSH A
              LDA A \1
              STA A \0
              PUL A
              STA A \1
              PUL A
            ENDC
          ENDC
        ENDC
        ENDM
```


RECURSIVE MACRO CALLS

Example :

```
* macro TABLE N generates a table of
* values, one per byte, from Ø to N
^
TABLE   MACR
        IFNE  \0
          TABLE  \0-1
        ENDC
        FCB   \0
        ENDM
```

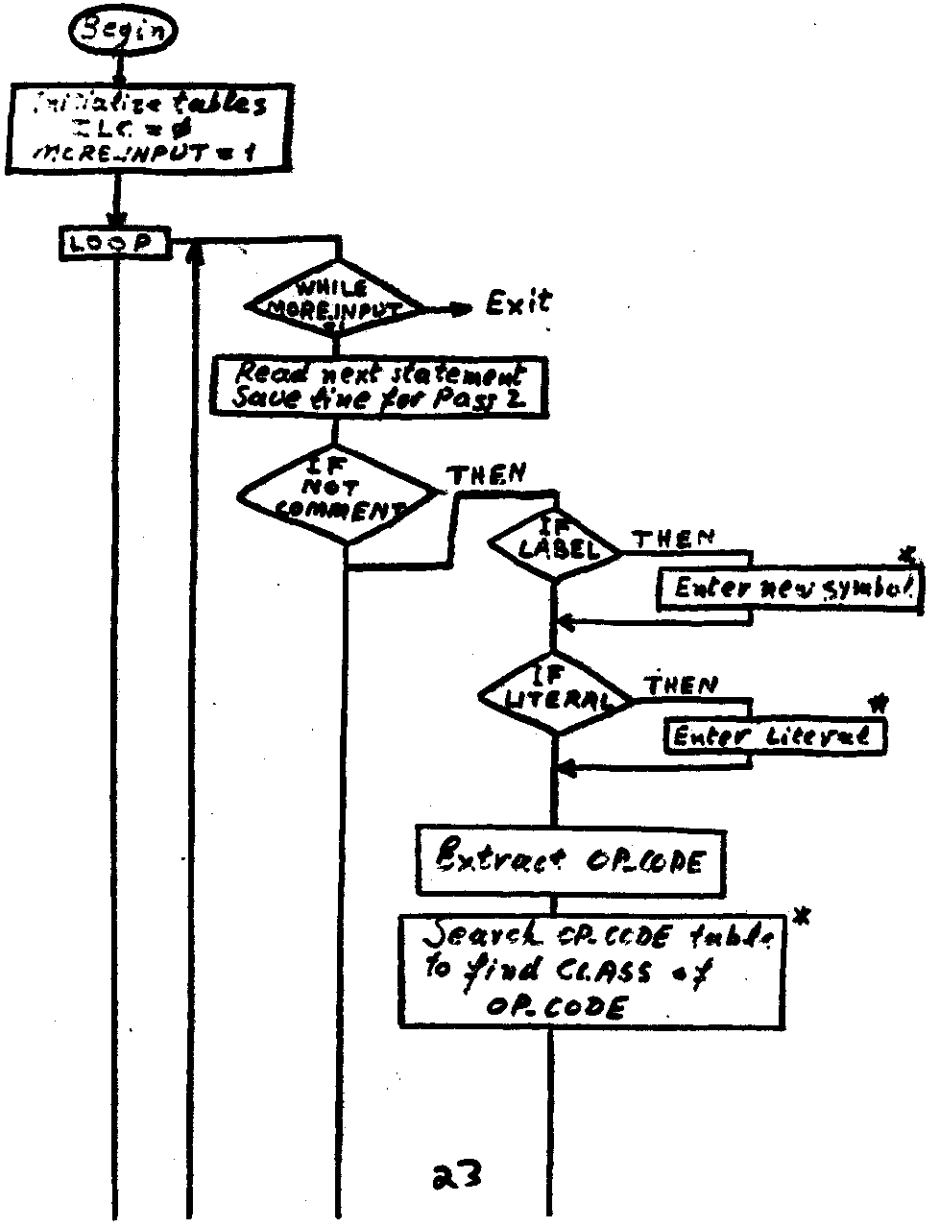
* macro expansion of TABLE 4

```
        TABLE 4
        ------
        TABLE 3
        FCB   4
        ------
        TABLE 2
        FCB   3
        FCB   4
        ------
        TABLE 1
        FCB   2
        FCB   3
        FCB   4
        ------
        TABLE 0
        FCB   1
        FCB   2
        FCB   3
        FCB   4
        ------
00      FCB   Ø
01      FCB   1
02      FCB   2
03      FCB   3
04      FCB   4
```

ASSEMBLY PROCCESS

- TWO-PASS ASSEMBLERS

PASS ONE



PASS-ONE (CONTINUED)

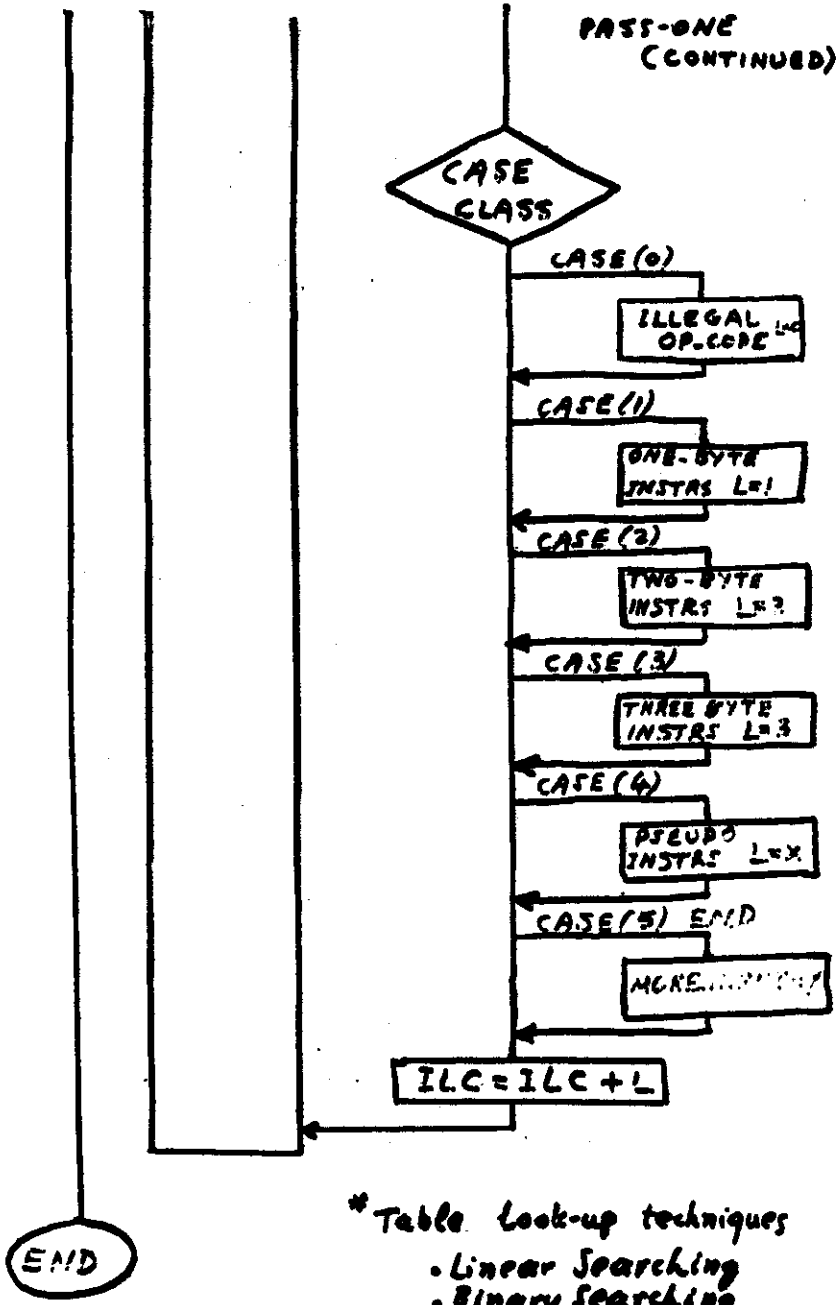


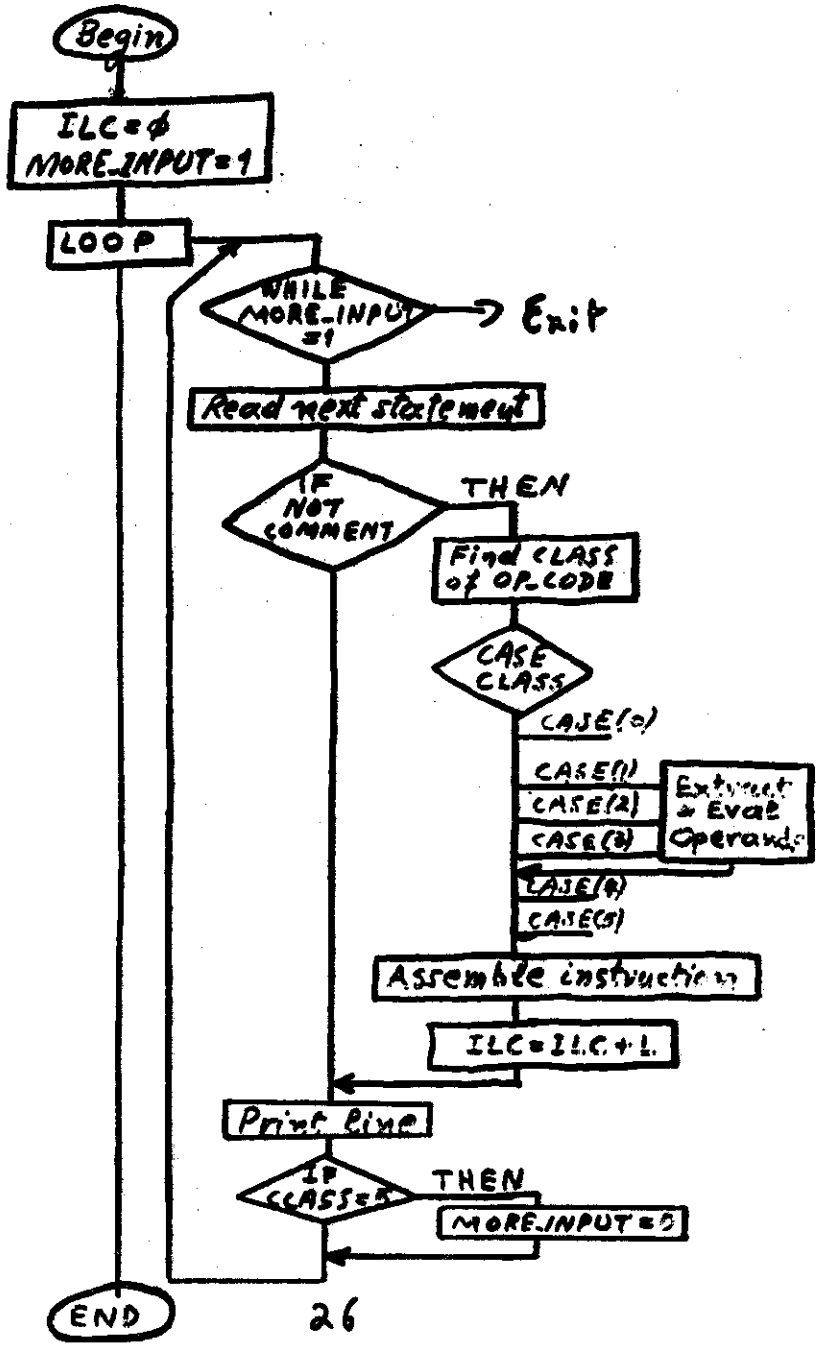
*Table Look-up techniques

- Linear Searching
- Binary Searching
- Hash Coding
- Sorting

PASS ONE

```
ILC = 0
MORE_INPUT = 1
Initialize_tables
WHILE (MORE_INPUT) = 1
   Read_next_line (LINE)
   Write_line_for_Pass2 (LINE)
   Is_this_a_comment (LINE, COM)
* COM IS SET TO 0 FOR COMMENT LINES AND 1 OTHERWISE *
   IF COM=0 THEN
      Check_for_label (LINE, LABEL)
      IF LABEL ¬= ' ' THEN
         Enter_new_symbol (LABEL, ILC) *SYMBOL TABLE*
      ENDIF
      Check_for_literal (LINE, LITERAL)
      IF LITERAL ¬= ' ' THEN
         Enter_literal (LITERAL) *LITERAL TABLE*
      ENDIF
* DETERMINE THE NATURE OF THE OPCODE *
      Extract_opcode (LINE, OPCODE)
      Search_opcode_table (OPCODE, CLASS, VALUE)
      CASE (CLASS)
        CASE(0): Illegal_opcode (LINE)
        CASE(1): Type_one (LINE, L)
        CASE(2): Type_two (LINE, L)
        CASE(3): Type_three (LINE, L)
        CASE(4): Pseudo_instruction (LINE, L)
        CASE(5): * END *
                 MORE_INPUT=0
                 Rewind_Pass2_input
      END CASE
      ILC = ILC+L
   ENDIF
ENDWHILE
END TYPE_ONE
```


PASS TWO



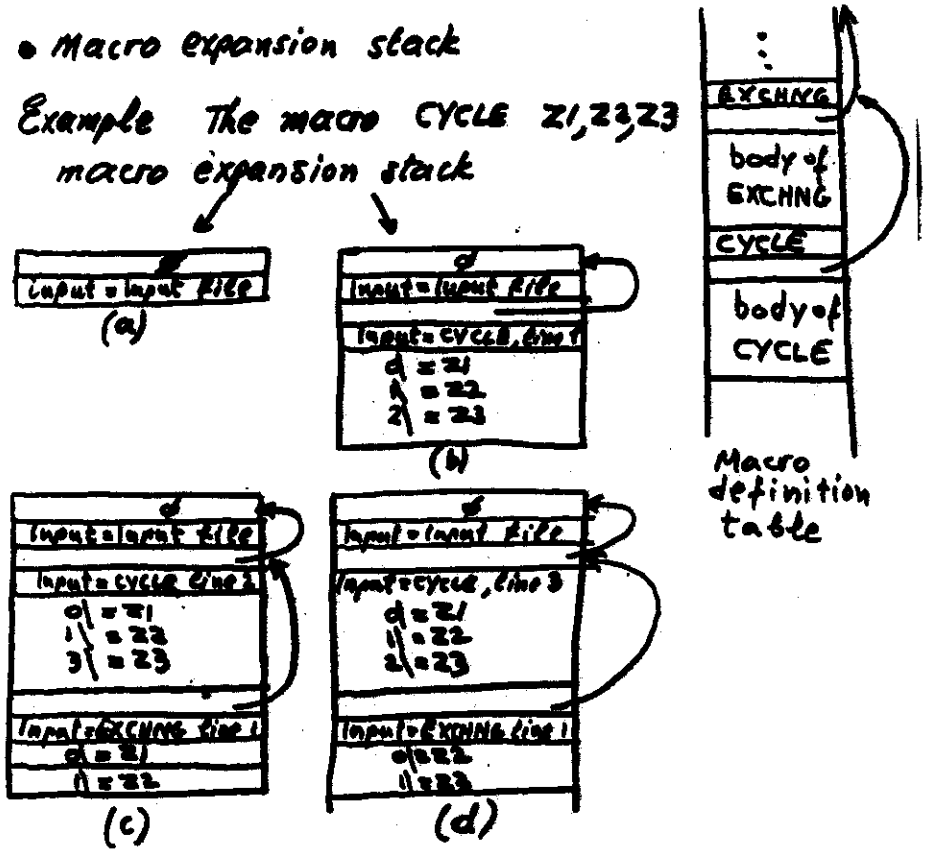
PASS TWO

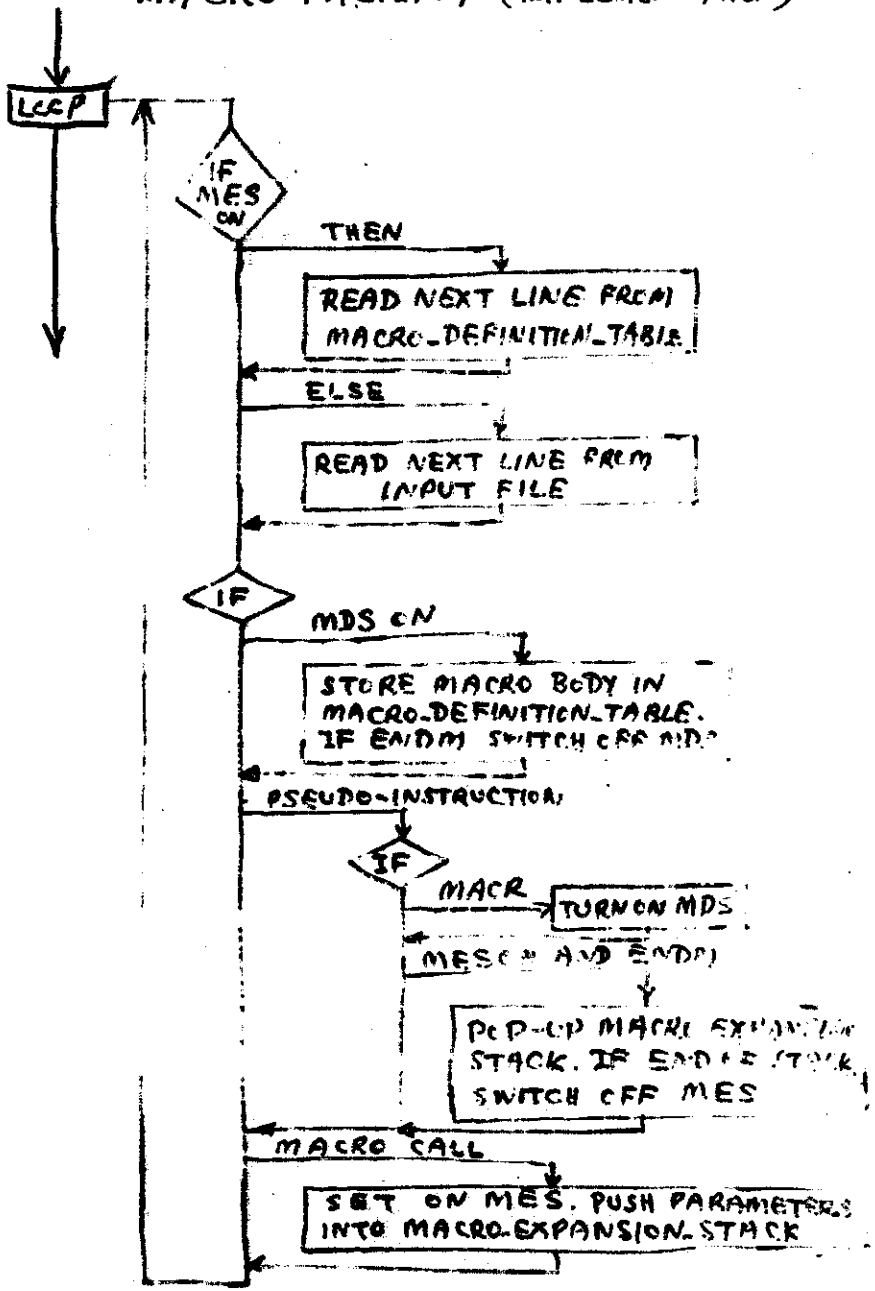
```
ILC=0
MORE_INPUT=1
DO WHILE (MORE_INPUT=1)
  Read_next_line (LINE)
  Is_this_a_comment (LINE, COM)
  IF COM=0 THEN
    Extract OP_CODE (LINE, OP_CODE)
    Search_OP_CODE_table (OP_CODE, CLASS, VALUE)
    CASE (CLASS)
      CASE(0) : ILLEGAL_OP_CODE
      CASE(1) : EVAL_1 (LINE, OPERANDS, L)
      CASE(2) : EVAL_2 (LINE, OPERANDS, L)
      CASE(3) : EVAL_3 (LINE, OPERANDS, L)
      CASE(4) : PSEUDO_INSTRUCTION
      CASE(5) : END
    ENDCASE
    Assemble_pieces (CODE, CLASS, VALUE, OPERANDS)
    Output_instruction (CODE)
    ILC=ILC+L
    ENDIF
    Print_listing (LINE, CODE, COM, ILC)
    IF CLASS=5 (END) THEN
      MORE_INPUT=0
    ENDIF
ENDDO_WHILE
END_PASS_TWO
```


IMPLEMENTATION OF A MACRO FACILITY IN AN ASSEMBLER

- macro definition table
- macro expansion stack

Example The macro CYCLE Z1,Z2,Z3





IF ... ELSE ... ENDIF

EQ
NE
GE
GT
LE
LT

Example

```
IF D,GE,#0          If D≥0
   JSR SQROOT         find √D
ELSE                else
   JSR ARGERR         error
ENDIF               endif
```

⇓

```
CMPD #0
BLT LABEL1
JSR SQROOT
BRA LABEL2
LABEL1  JSR ARGERR
LABEL2  EQU *
```

IFTST ....ELSE ....ENDIF

EQ
NE
GE
LT

Example

```
IFTST D,NE,0        if D≠0
  JSR RECIP         find the reciprocal I/D
ELSE                else
ENDIF               endif
```

IFCC ....ELSE....ENDIF test the CC

Example

```
ROLA
IFCC GT
  JSR DOIT
ENDIF
```

WHILE .... ENDWH

```
WHILE A,GT,#0       Do while (A>0)
  LSL B               shift left B
  DEC A               decrement A
ENDWH               ENDDOWHILE
```

REPEAT .... UNTIL

```
REPEAT              REPEAT
  LSL B               shift left B
  DEC A               decrement A
UNTIL A,LE,#0       UNTIL (A≤0)
```


LOADERS & LINKERS

- LOAD OBJECT INTO μC RAM

- - - - - - - - - - - - - - - - -

- CONVERSION OF RELOCATABLE CODE INTO LOADABLE CODE

- ESTABLISHMENT OF LINKAGES BETWHEEN OBJECT MODULES (LINKAGE EDITING)

LOADERS

BINARY (ABSOLUTE) LOADER

- loads a single program module in absolute binary form

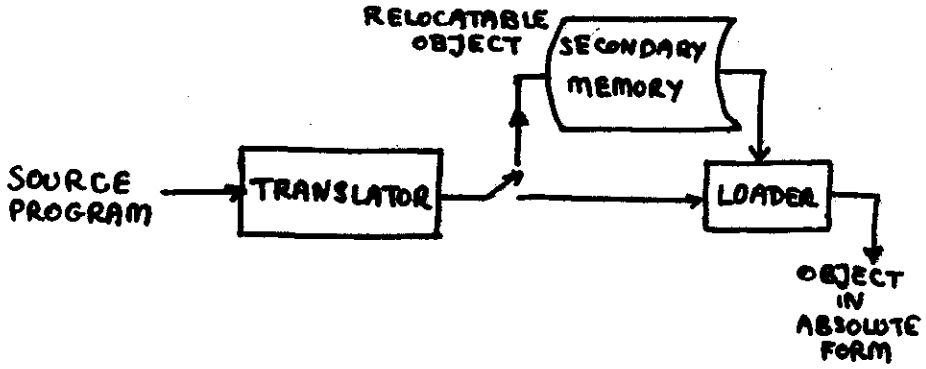
RELOCATING LOADER

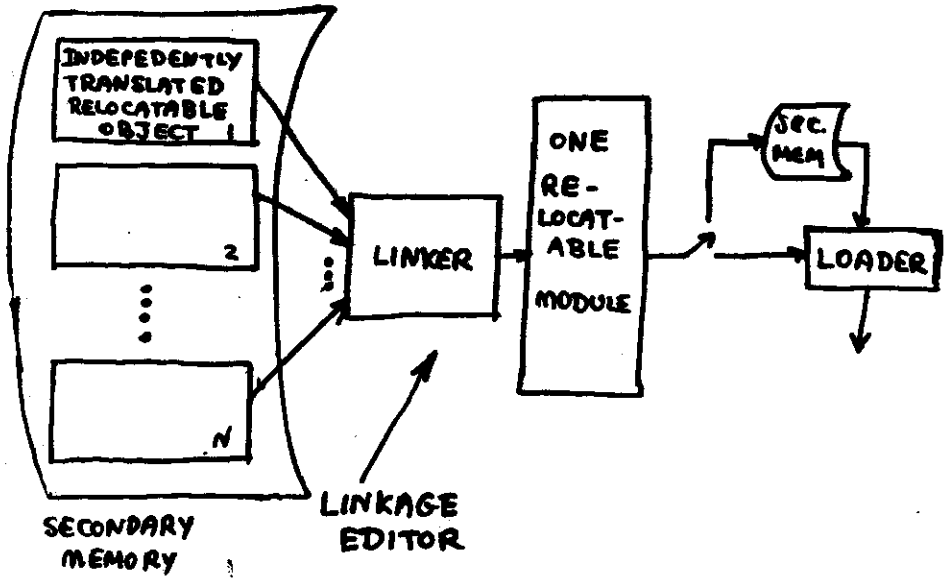
- Relocatable program
  - address fields have been translated relative to zero
  - relocation information indicates which address fields must be relocated by the loader

RELOCATION

				PIAD	EQU	$4000	ORG $2000		
00:	B6	1000	LOAD	LDAA	ALPHA		2000	B6	3000
03:	B7	1001		STAA	BETA		2003	B7	3001
06:	7E	0100		JMP	READ			7E	2100
⋮				⋮					
000	F6	4000	READ	LDAB	PIAD		2100	F6	4000
03:	7E	0000		JMP	LOAD		2103	7E	2000
				⋮				⋮	
100	00		ALPHA	RMB	1		3000	00	
101	00		BETA	RMB	2		3001	00	

BASIC LANGUAGE TRANSFORMATION PROCESS





LINKING AFTER TRANSLATION BUT BEFORE EXECUTION

LINKER

- IT LINKS TOGETHER PROGRAM MODULES INTO A COMPOSITE PROGRAM
- LINKING MAY BE DONE AT SEVEN DIFFERENT TIME INSTANCES

① AT SOURCE PROGRAM CODING TIME

② AFTER CODING BUT BEFORE TRANSLATION

③ AT TRANSLATION TIME

④ AFTER TRANSLATION BUT BEFORE LOADING

⑤ AT LOADING TIME

⑥ AFTER LOADING BUT BEFORE EXECUTION

⑦ AT EXECUTION TIME

① MEANS MANUAL LINKING

②③ IMPLY RE-TRANSLATION

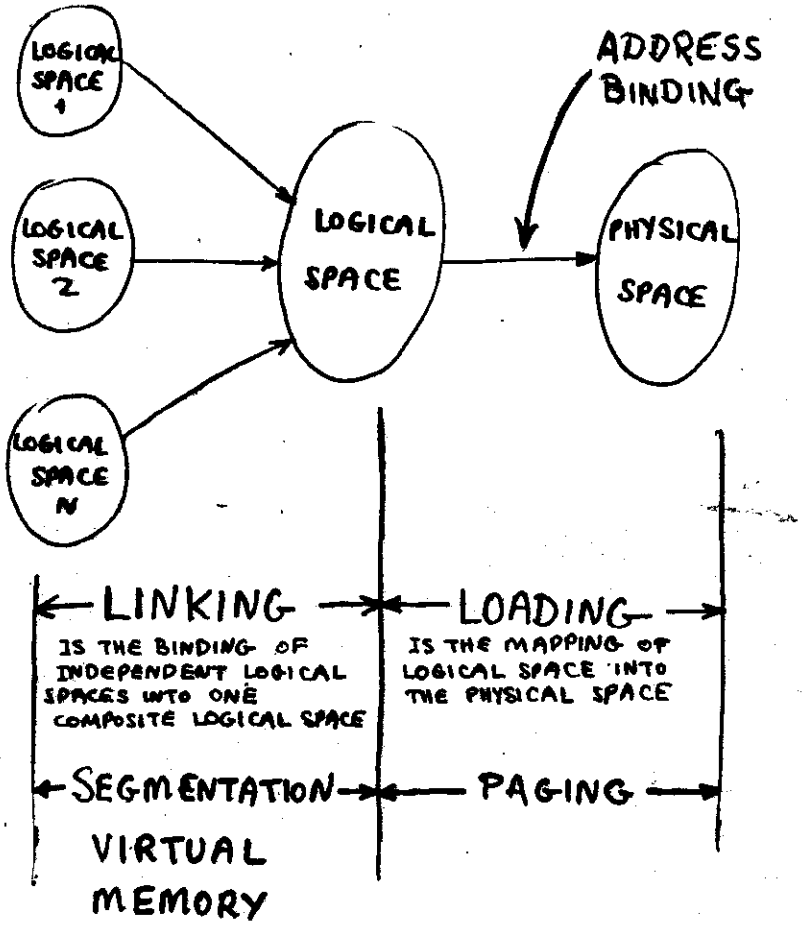
④ IMPLIES A LINKAGE EDITOR (LINKER)

⑤ IMPLIES A LINKING LOADER

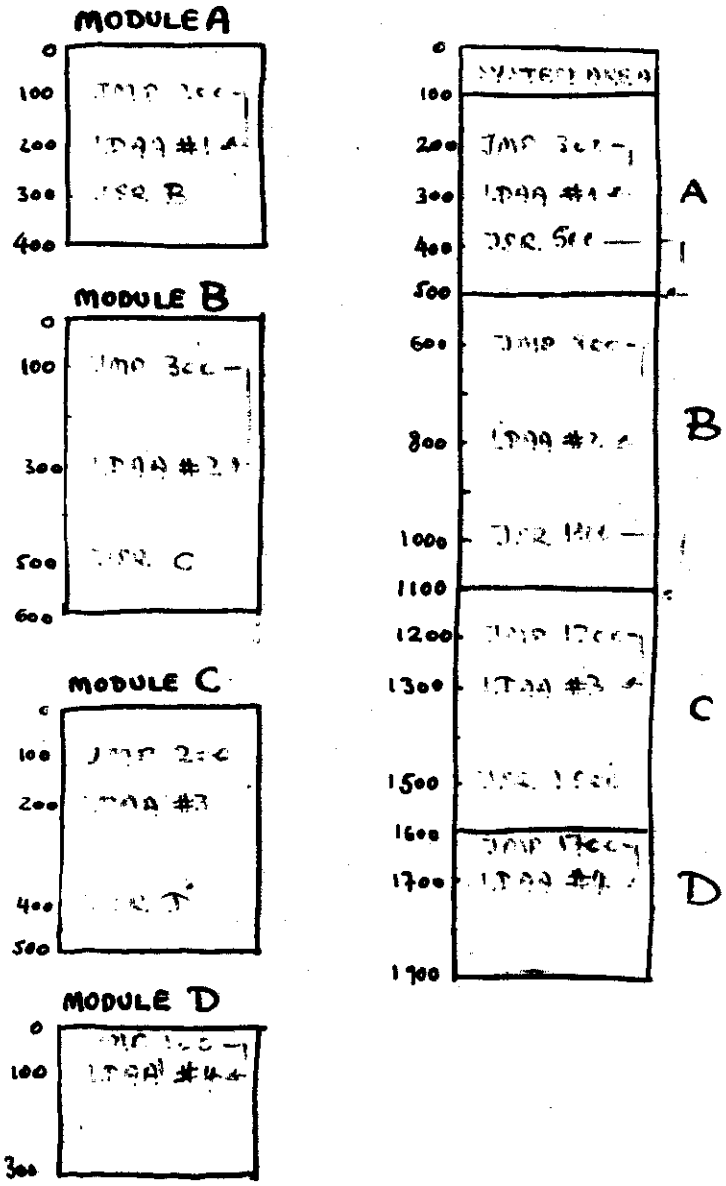
⑥ IMPLIES TO KEEP RESIDENT A LARGE NUMBER OF SUBPROGRAMS

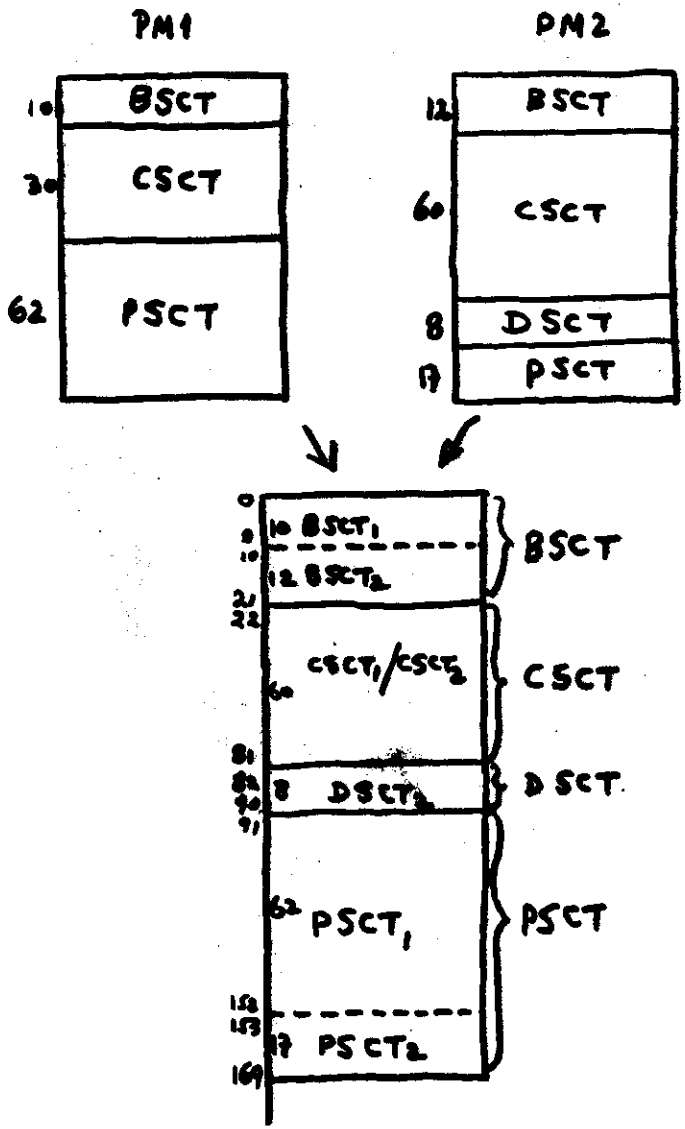
⑦ IMPLIES DYNAMIC LINKING

AN ANOTHER VIEW OF LINKING AND LOADING



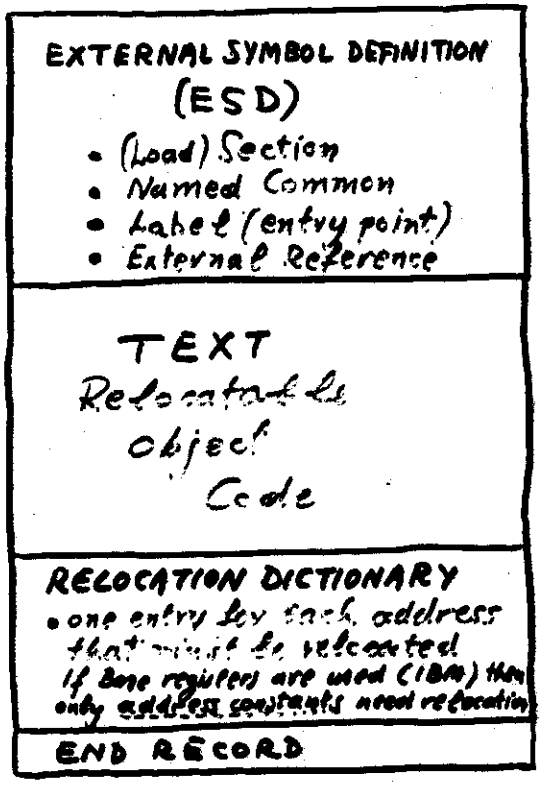
EXAMPLE OF LINKING



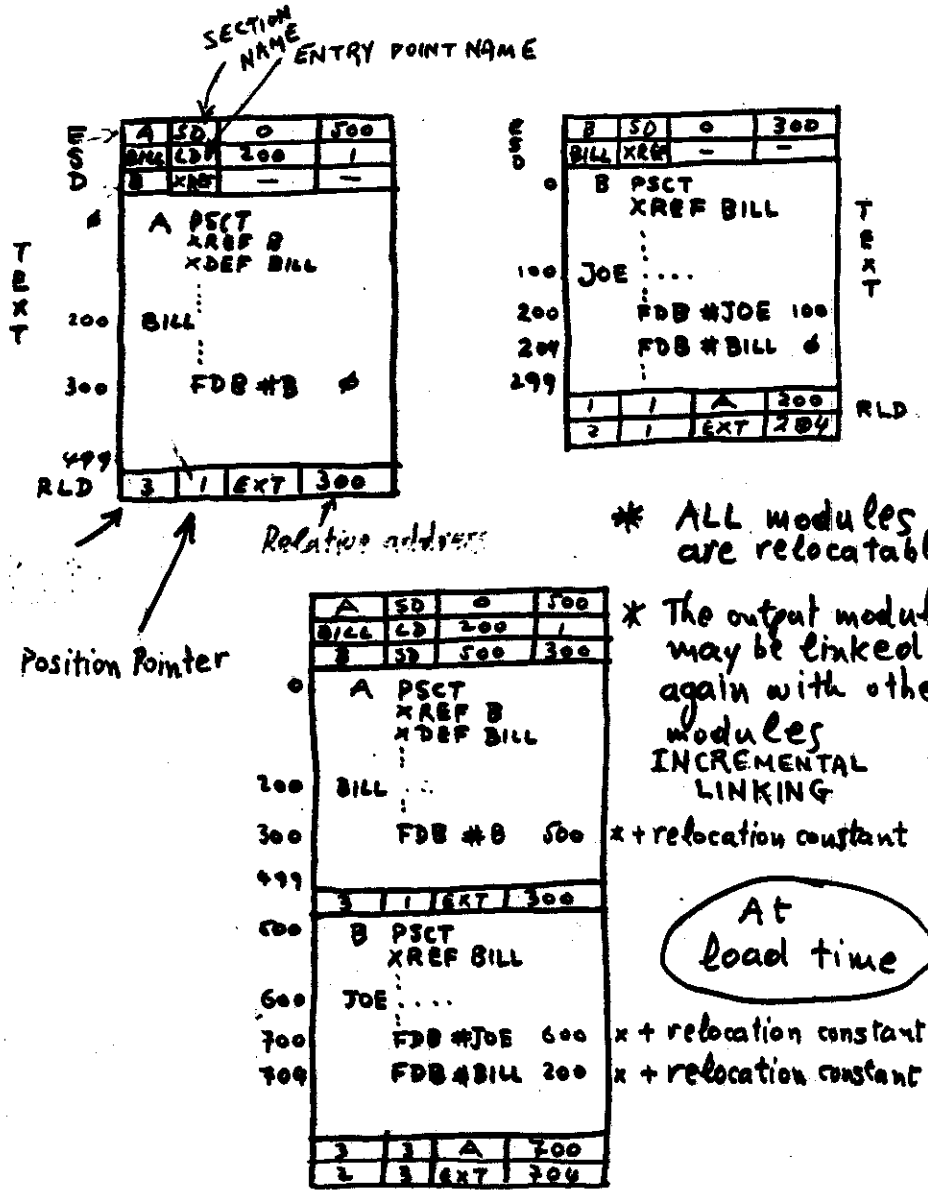


Example of load map for two modules

FORMAT OF RELOCATABLE OBJECT MODULE



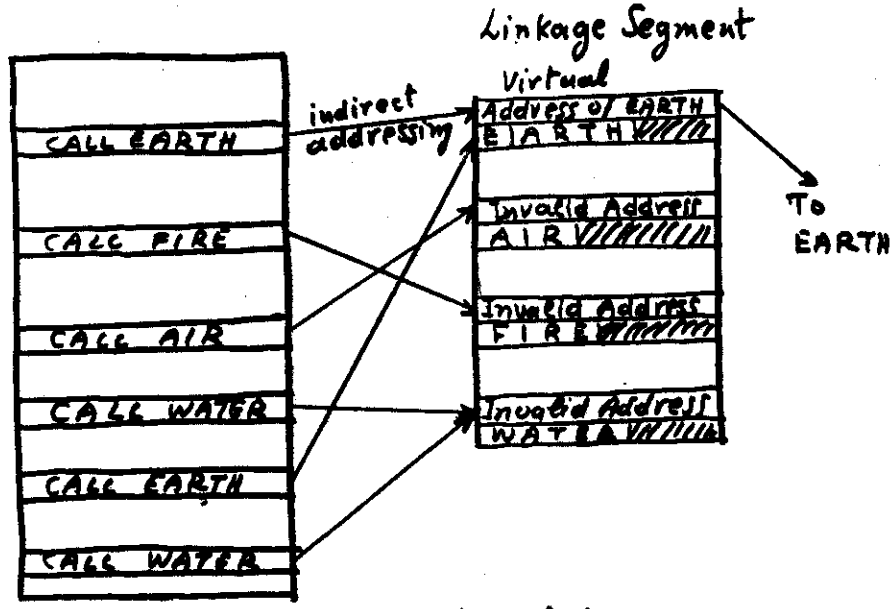
External Symbols
- External names
  - Section names
  - Entry points XDEF
- External references XREF



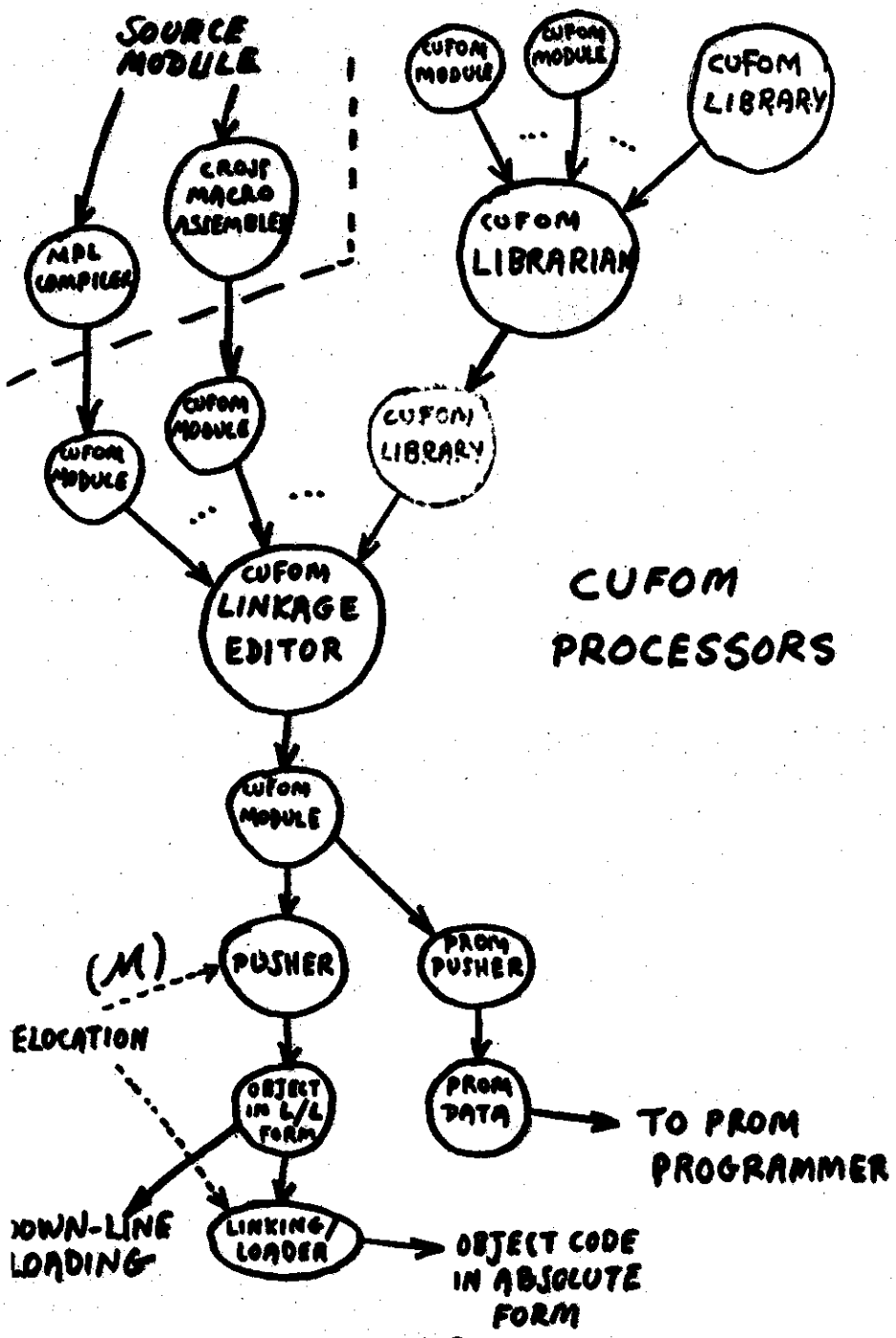
Linking example (base registers are assumed)

DYNAMIC LINKING

- Linking at execution time. Each module (procedure is linked at the time it is first called.
- Linkage Segment



Before linking a trap occurs when first accessing Invalid Address



CUFOM PROCESSORS