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

INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
4100 TRIESTE (ITALY) - P.O.B. 586 - MIRAMARE - STRADA COSTIERA 11 - TELEPHONES: 224281/2/3/4/5/6
CABLE: CENTRATOM - TELEX 460392-1

SMR/90 - 32

COLLEGE ON MICROPROCESSORS:

TECHNOLOGY AND APPLICATIONS IN PHYSICS

7 September - 2 October 1981

PROGRAMMING TOOLS AND TECHNIQUES

C. HALATSIS
Physics & Mathematics Dept
Digital Systems and Computers Lab.
University of Thessaloniki
Thessaloniki
Greece

These are preliminary lecture notes, intended only for distribution to participants. Missing or extra copies are available from Room 230.

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

- General purpose text editors
  "CONTEXT FREE"

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

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

{MOVE, FIND} EDITING WINDOW
EDITOR POINTER

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

Commands of the DAN Editor

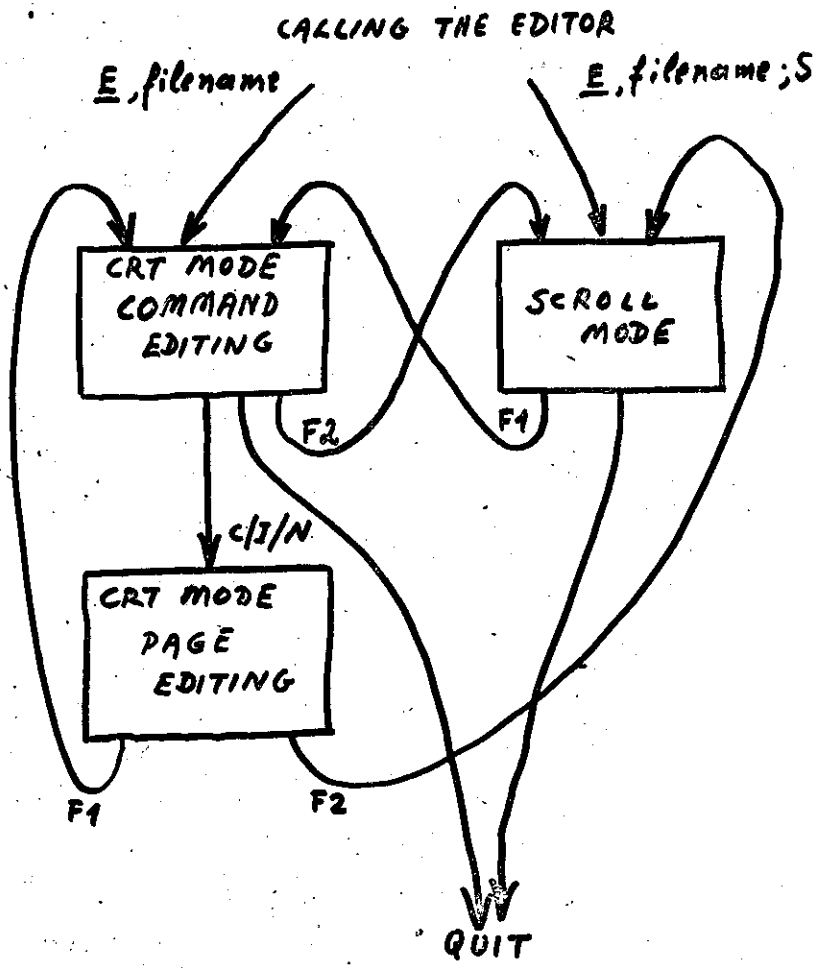
Call it DAN, lfn

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]

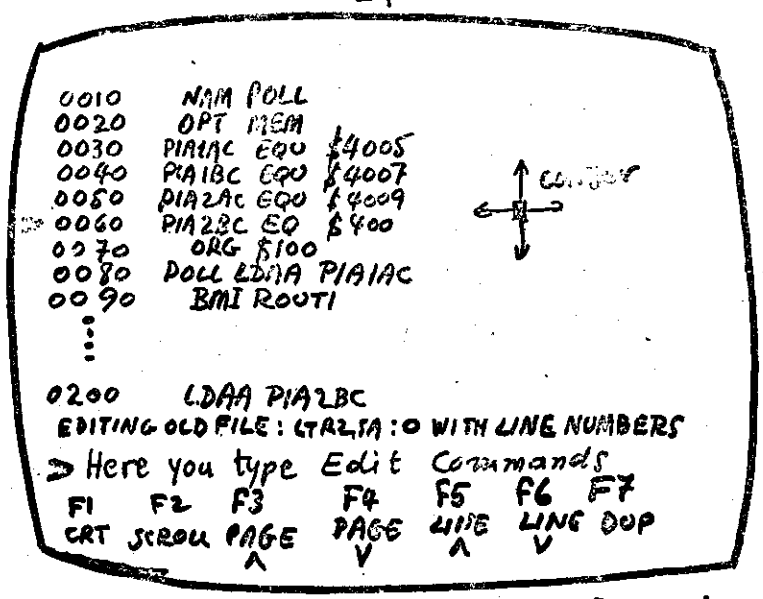
/text1/=/text2/[,line range][,VETO][,NEXT]
[,text condition][,column range][,UNIT]

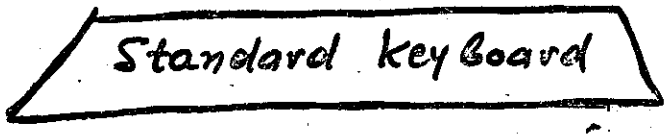
text selection =/text/[,column range][,text condition][,UNIT]

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





Keys used when in Page Mode

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

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

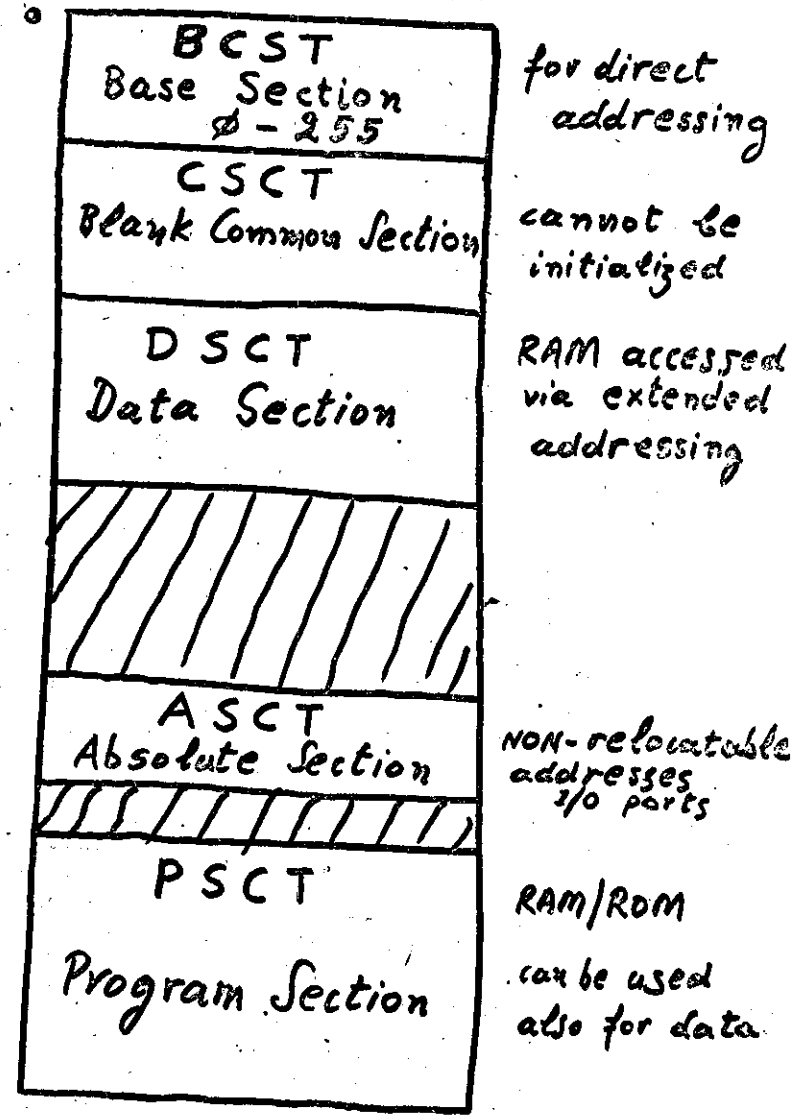
ASSEMBLY LANGUAGE

- 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 ⇔ MACHINE INSTRUCTIONS
- PSEUDO INSTRUCTIONS ⇒
  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



• CHARACTER SET

upper case letters A to Z
digits 0-9
arithmetic operators + - * /
two-char operators !^ [XR] !> [LSR] !< [LSL] !X [OR] !. [AND] !+ [XOR] !R [ROR] !L [ROL]
parentheses in expressions ( )

• PREFIXES FOR CONSTANTS & ADDRESSING MODES

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

• ASSEMBLY CONTROL

NAM, OPT {REL, LOAD, OBJ, MEM}, ORG, {ASCT, BSCT, CSCT, DSCT, PSCT}, COMM, [illegible]

• 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, CREF, [illegible], [illegible], [illegible], [illegible], [illegible], [illegible], [illegible], [illegible], [illegible], [illegible]}

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)

- a macro is in effect 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
      PSHA        stack ← A
      TBA           A ← B
      PULB          B ← stack
      ENDM

* macro call
      SWAP

* macro expansion
      SWAP
36    PSHA
17    TBA
33    PULB
```

MACROS with parameters

Example

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

CHANGE  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
        CHANGE  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
        CHANGE  W,T
        PSH A
        LDA A  W
        PSH A
        LDA A  T
        STA A  W
        PUL A
        STA A  T
        PUL A
```


NESTED MACRO CALLS

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

Example

```
* 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
        |  PSH A
        |  LDA A  X1
        |  PSH A
        |  LDA A  X2
        |  STA A  X1
        |  PUL A
        |  STA A  X2
        (  PUL A
        (  EXCHNG X2,X3
        |  PSH A
        |  LDA A  X2
        |  PSH A
        |  LDA A  X3
        |  STA A  X2
        |  PUL A
        |  STA A  X3
        |  PUL A
```

NESTED MACRO DEFINITIONS

when a macro definition is contained into the definition of another macro

Example

```
* macro EXCHNG Y,Z exchanges Y⇄Z, where
* Y,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
```


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
```

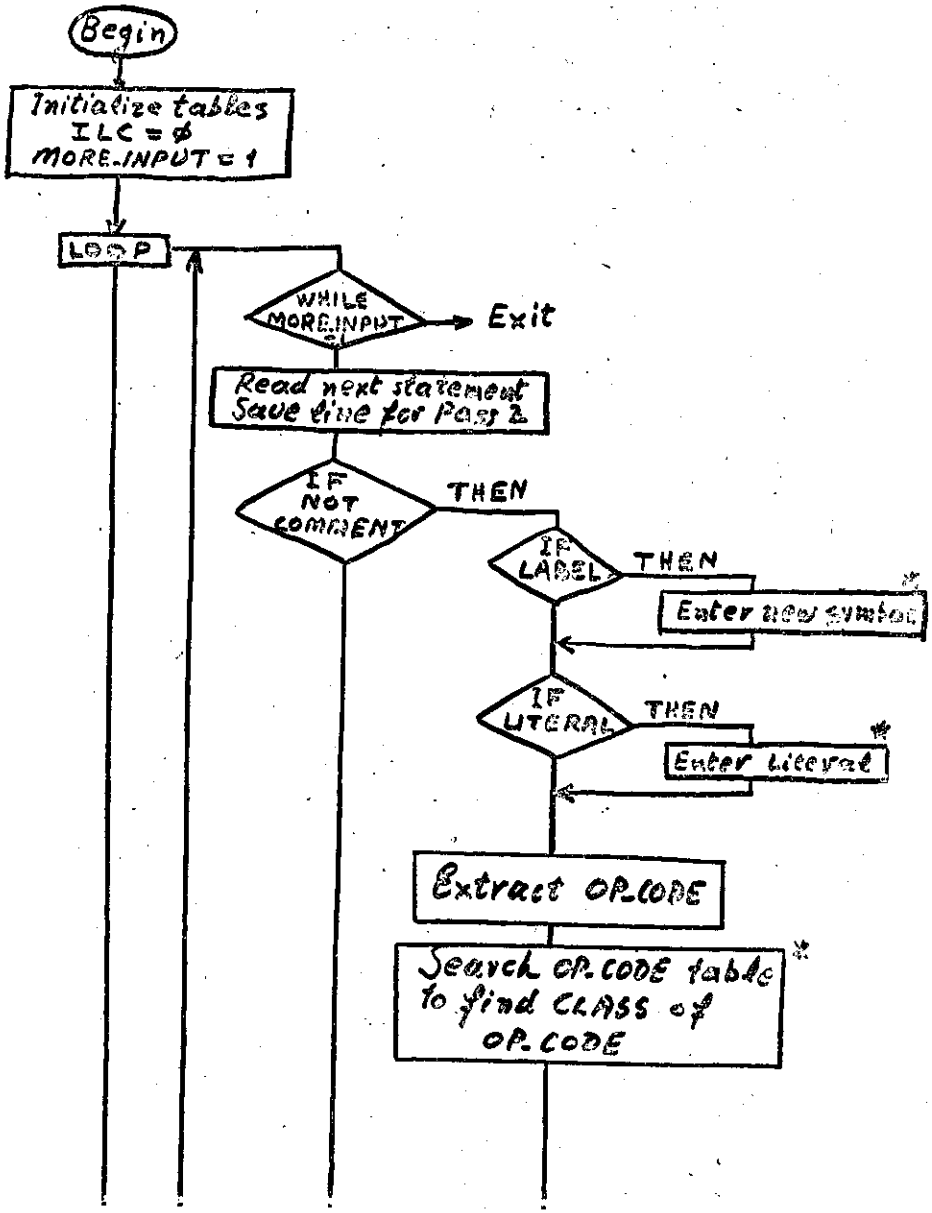
*macro expansion of TABLE 4

```
        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
```

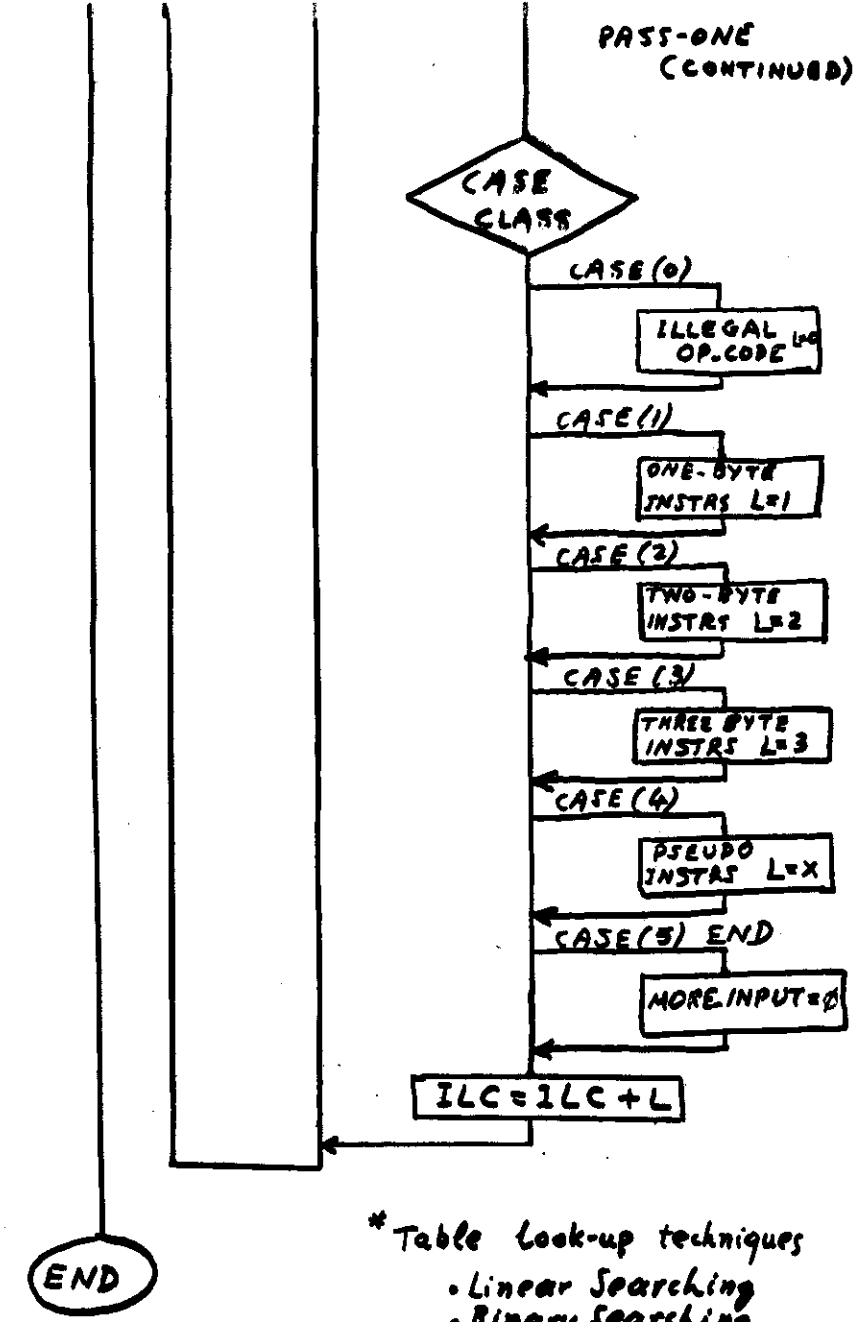
ASSEMBLY PROCCESS

• TWO-PASS ASSEMBLERS

PASS ONE



PASS-ONE (CONTINUED)



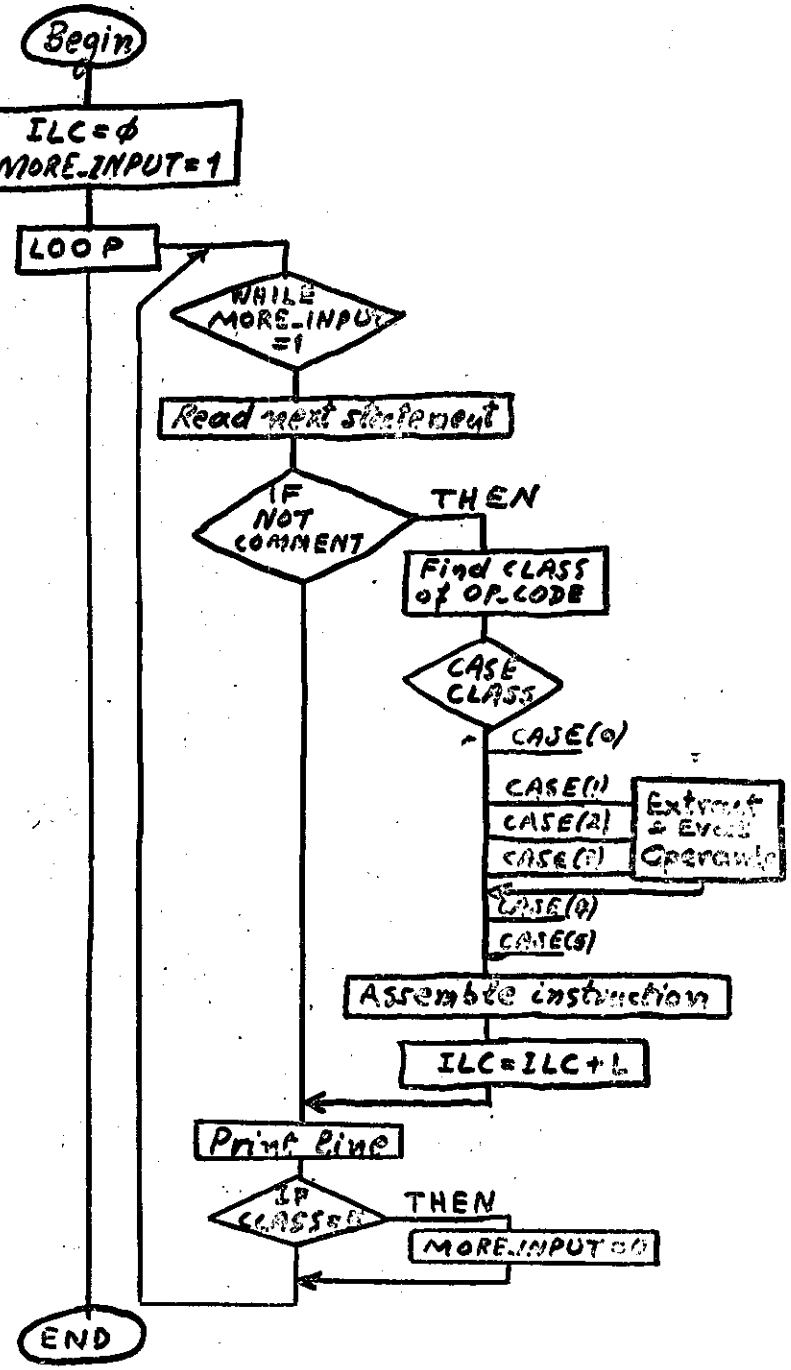
* Table look-up techniques

- Linear Searching
- Binary Searching
- Hash Coding
- Sorting

PASS ONE

```
ILC=0
MORE_INPUT=1
Initialize tables
DO WHILE (MORE_INPUT=1)
    Read_next_line(line)
    Save_line_for_Pass2(line)
    Is_this_a_comment(line, COM)
/* COM is set to 1 for comment lines, 0 otherwise
    IF COM=0 THEN
        Check_for_label(line, LABEL)
        IF LABEL THEN
            Enter_new_symbol(LABEL, ILC)
        ENDIF
        Check_for_literal(line, LITERAL)
        IF LITERAL THEN
            Enter_literal(LITERAL)
        ENDIF
        Extract OP_CODE(line, op_code)
        Search_OP_CODE_table(OP_CODE, CLASS)
        CASE (CLASS)
          CASE(0): ILLEGAL_OP_CODE ;
          CASE(1): One_byte_instructions(L);
          CASE(2): Two_byte_instructions(L);
          CASE(3): Three_byte_instructions(L);
          CASE(4): Pseudo_instruction;
          CASE(5): END
                   Move_input=0
                   Rewind input;
        END CASE
    ENDIF
ENDWHILE
END_PASS_ONE
```


PASS TWO



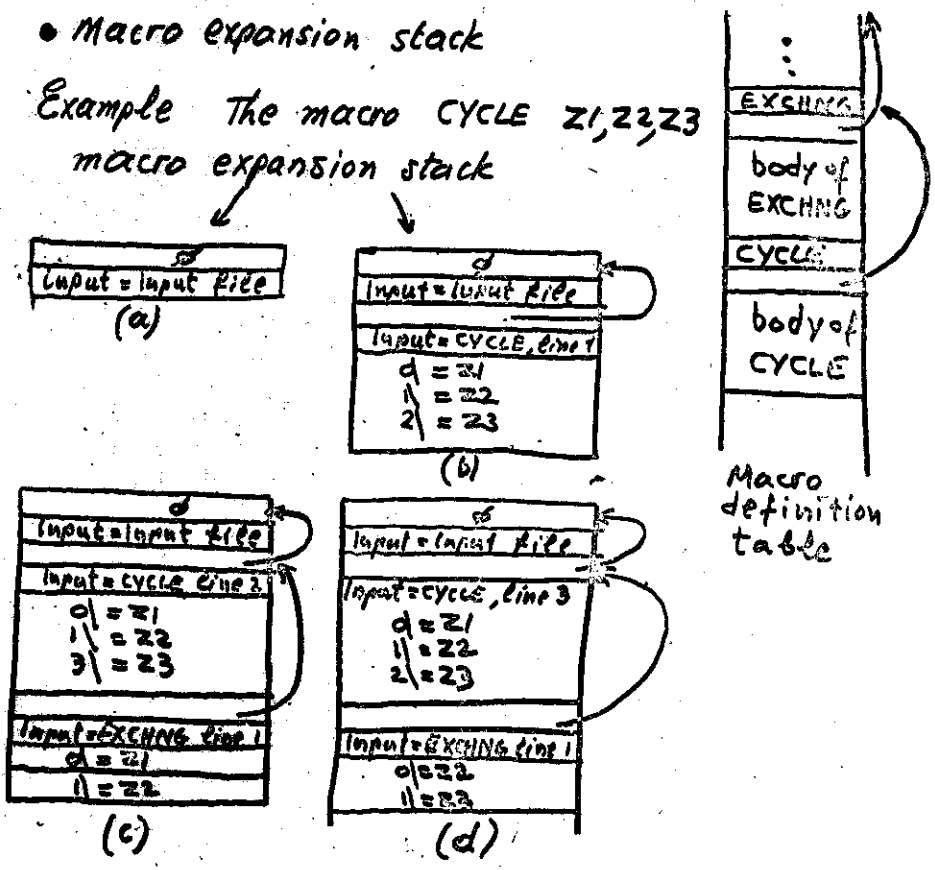
PASS TWO

```
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
```

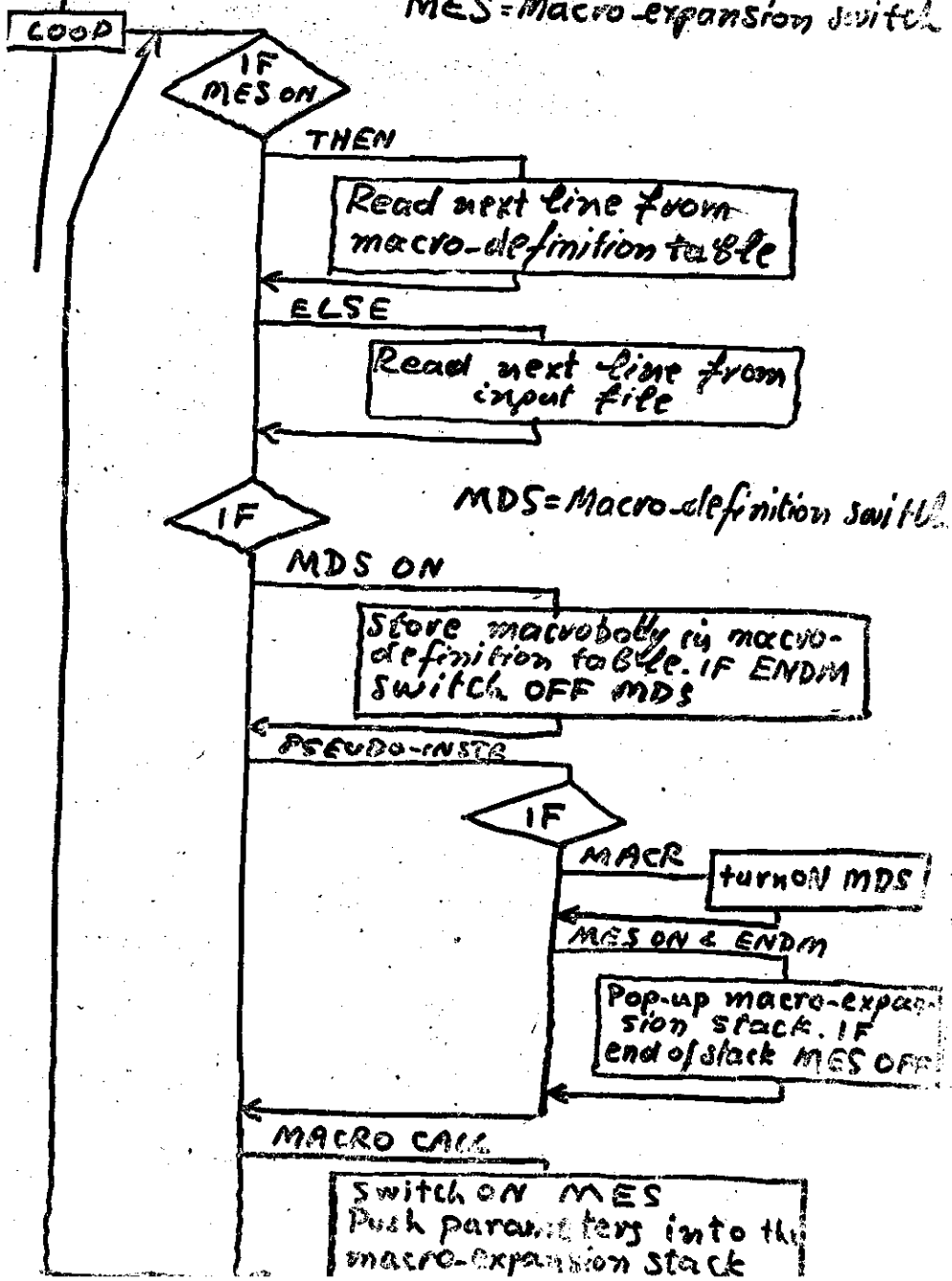

IMPLEMENTATION OF A MACRO FACILITY IN AN ASSEMBLER

- macro definition table
- Macro expansion stack

Example The macro CYCLE Z1,Z2,Z3



MACRO FACILITY



HLL's FOR FIXED INSTRUCTION SET μP's

- STRUCTURED ASSEMBLERS
  PLZ-ASM, M 6809
- MACHINE DEPENDENT HLL's
  Smal, BSAL, Mistral, PL/65
- μP-ORIENTED HLL's
  MPL, PLuS, PL/M, PLZ-SYS
- SYSTEM LANGUAGES
  BASIC, PASCAL, RTL/2, C
- APPLICATION ORIENDED HLL's
  APL, FORTRAN, COBOL

Structuring MACROS of the M 6809 Macro-Assembler

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 1/D
ELSE                else
ENDIF               endif
```

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

Example

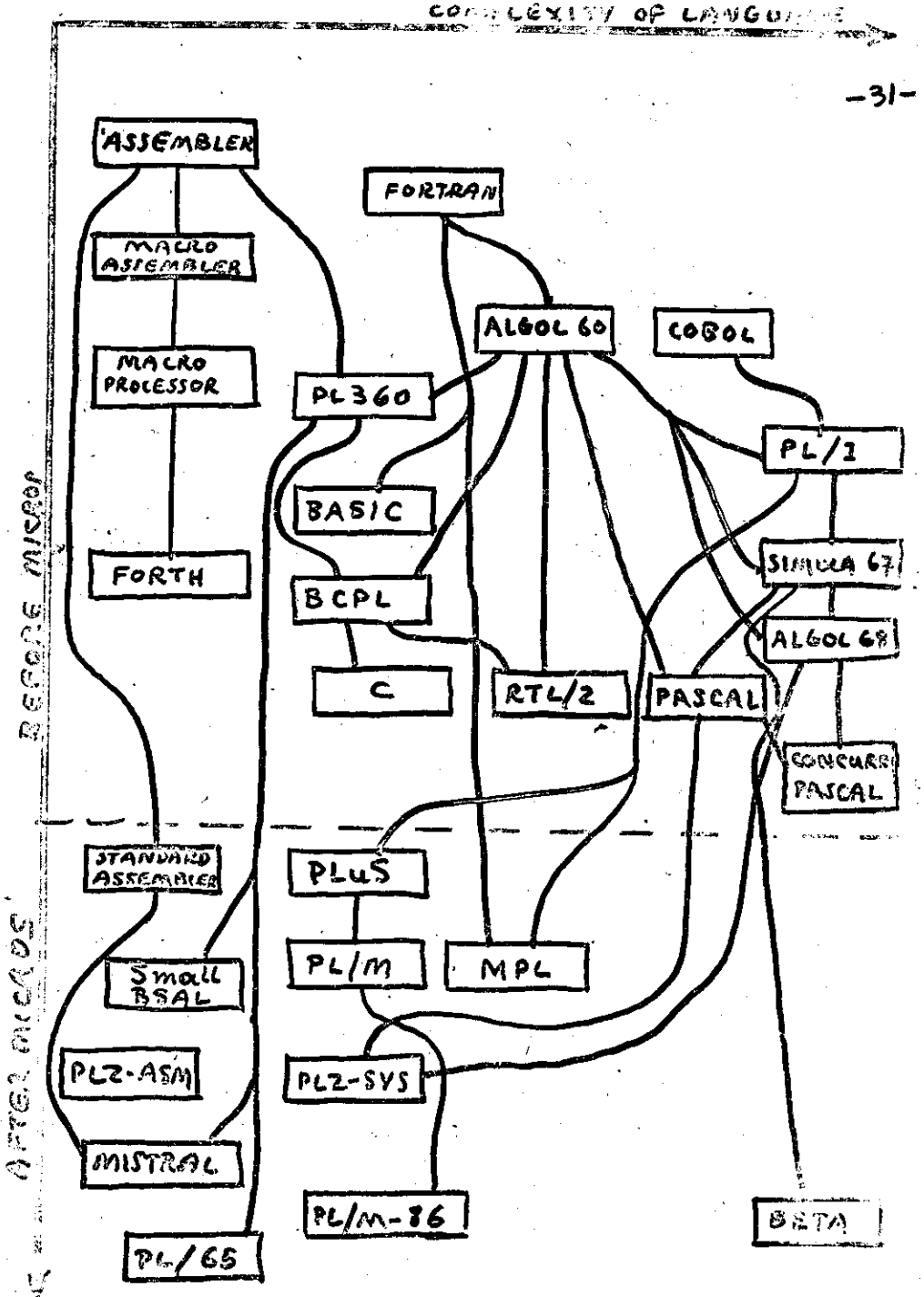
```
ROLA
IFCC GT
   JSR DOIT
ENDIF
```

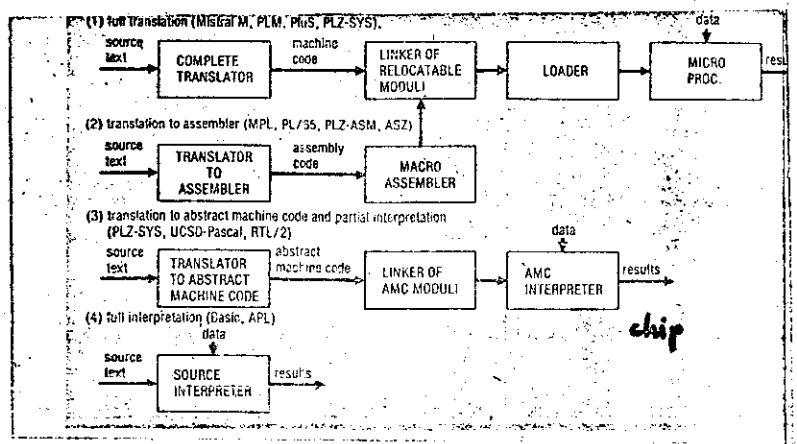
WHILE .... ENDWH

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

REPEAT .... UNTIL

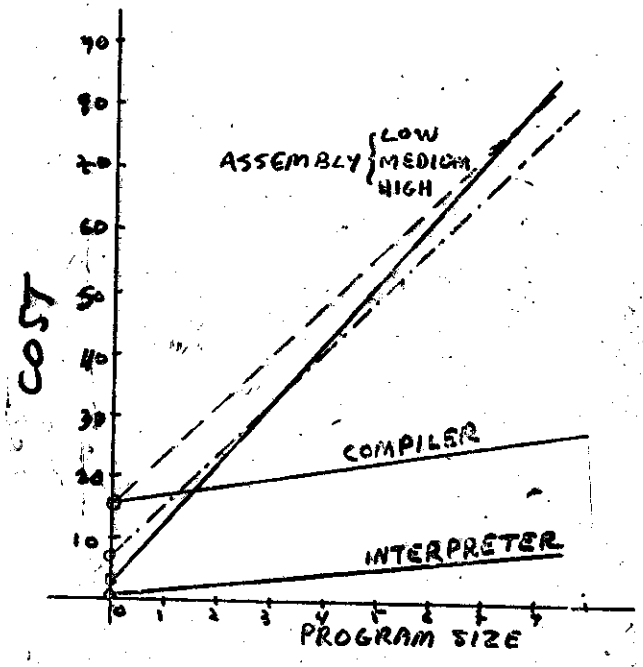
```
REPEAT              REPEAT
   LSL B              shift left B
   DEC A              decrement A
UNTIL A,LE,#0       UNTIL (A≤0)
```





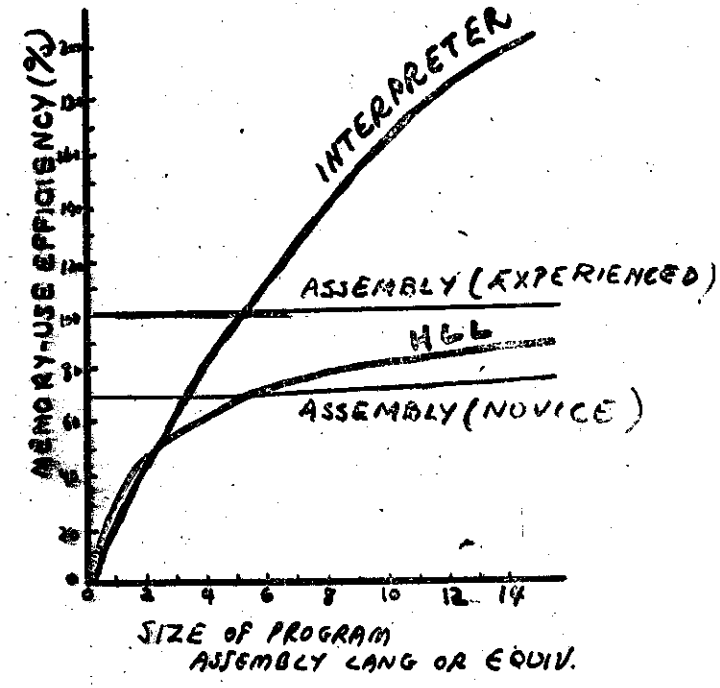
Spectrum of language translation options and their influence on program execution, program preparation and portability.

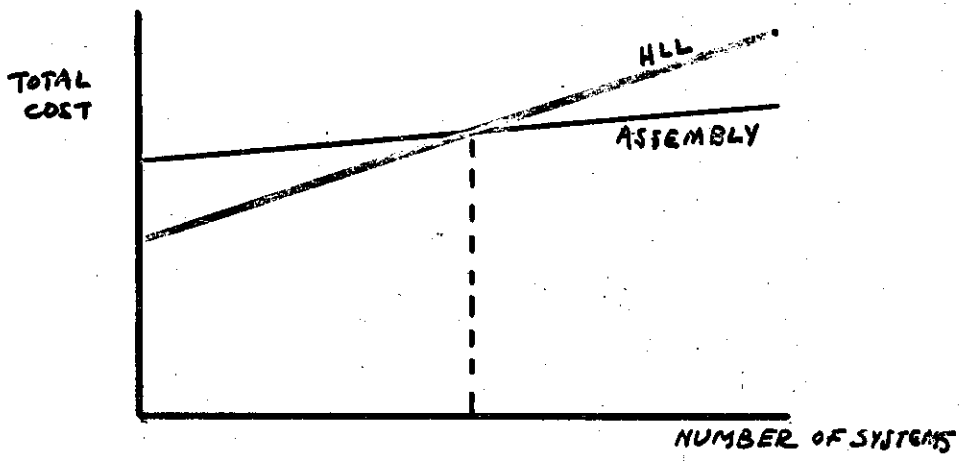
		COMPLETE TRANSLATION	TRANSLATION TO ASSEMBLER	TRANSLATION TO AMC AND AMC INTERPRETATION	S
program execution	EXECUTION SPEED	1	1	2	3
	SIZE OF OBJECT PROGRAM	3	3	2	1
	SIZE OF RUN-TIME SYSTEM	1	1	2	3
	RUN-TIME DIAGNOSTIC AND DEBUGGING	3	3	2	1
program preparation	OVERALL SIZE OF PROGRAM PROCESSING PACKAGE	3	4 (includes assembler)	2	1
	TIME TO PREPARE A PROGRAM FOR EXECUTION	3	4	2	1
	DIAGNOSTICS AND INTERACTIVITY	2	3	3	1
	MEMORY SIZE TO COMPILE	3	2	1	—
portability	PORTABILITY TO OTHER MACHINE	3 (good with retargeting techniques)	2	1	2



Start-up costs and programming productivity/cost

Programming memory-use efficiency





Crossover Point

HAND CODE

ADVANTAGES

- RELATIVELY FAST FOR SMALL PROGRAMS
- NO ASSEMBLY NEEDED

DISADVANTAGES

- SLOW AND TEDIOUS FOR LARGER PROGRAMS
- VERY DIFFICULT TO MODIFY PROGRAMS
- ERROR-PRONE

WHEN TO USE

- FOR VERY SMALL JOBS
- WHEN NO ASSEMBLER IS READILY ACCESSIBLE

Pros and cons of programming methods

ASSEMBLY LANGUAGE

ADVANTAGES

- SYMBOLIC REFERENCES
- EASIER TO MODIFY
- EASIER TO READ
- PARAMETERIZED VALUES (TABLE SIZES etc)
- ERROR CHECKING

DISADVANTAGES

- ASSEMBLER SYSTEM REQUIRED
- MUST LEARN FORMATS & RULES

WHEN TO USE

- FOR TIME-CRITICAL PROGRAMMING

MACROS (ASSEMBLY)

ADVANTAGES

- REPEATED SMALL GROUPS OF INSTRUCTIONS REPLACED BY ONE MACRO
- IN EFFCT NEW INSTRUCTIONS CAN BE CREATED

DISADVANTAGES

- MACROS ARE NOT SUBROUTINES

WHEN TO USE

- WHEN SUBROUTINES ARE NOT WORTHY
- TO CREATE HIGHER INSTRUCTION SET

HIGH LEVEL LANGUAGE

ADVANTAGES

- BETTER CONTROL OF SOFTWARE
- REDUCED PROGRAMMING COST
- FASTER PROGRAMMING
- SELF-DOCUMENTING
- EASIER MAINTENANCE
- TRANSFERABILITY

DISADVANTAGES

- BIGGER PROGRAMS - COMPILER EFFICIENCY
- COST OF COMPILATION
- PROGRAMMING EXPIRIENCE IS REQUIRED FOR GOOD RESULTS

WHEN TO USE

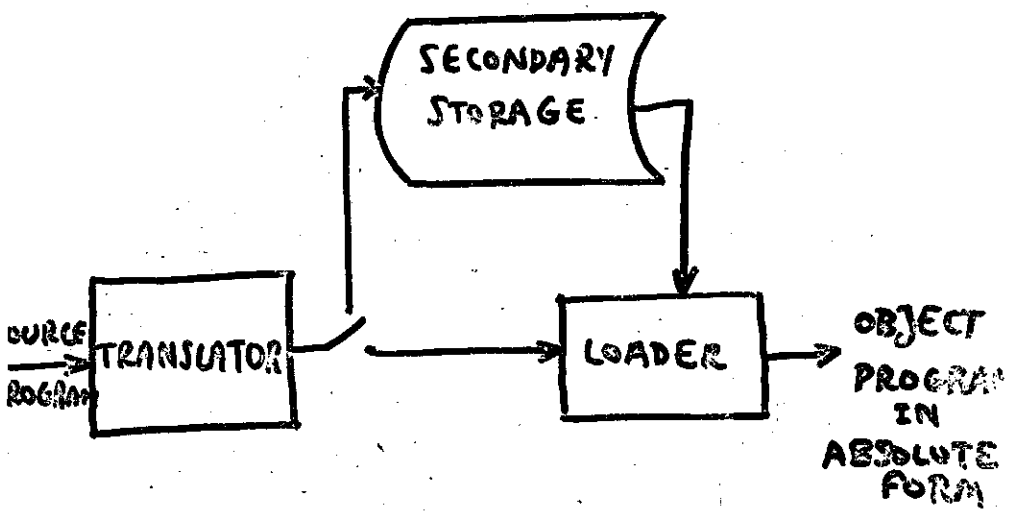
- FOR LARGER PROGRAMS
- FOR LOW-VOLUME PRODUCTS

LOADERS & LINKERS

- LOAD OBJECT INTO µC RAM

- CONVERSION OF RELOCATABLE CODE INTO LOADABLE CODE

- ESTABLISHMENT OF LINKAGES BETWHEEN OBJECT MODULES (LINKAGE EDITING)



Basic Language transformation process

LOADERS

BINARY (ABSOLUTE) LOADER

- loads a single program module in absolute binary form

• RELOCATING LOADER

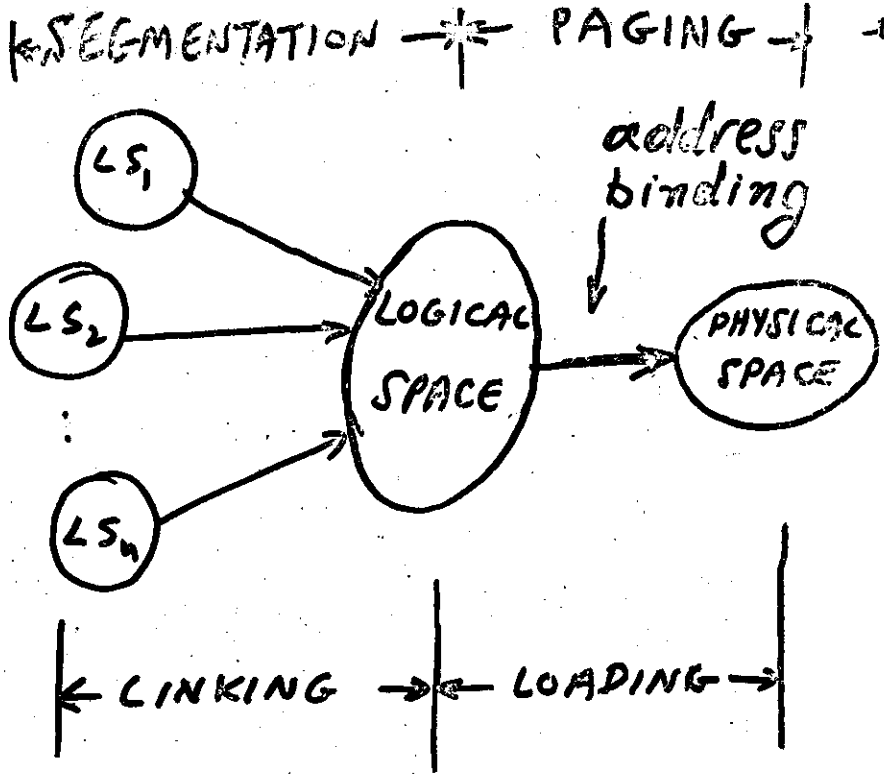
- Relocatable program
  - address fields have been translated relative to zero
  - relocation information indicates which address fields must be relocated by the loader

LINKER

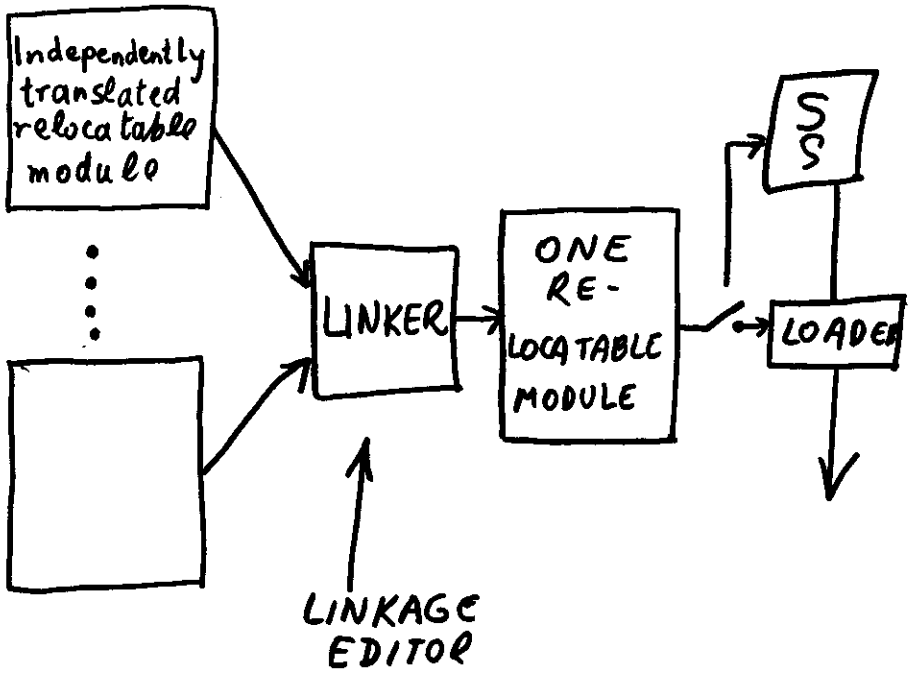
- links together program modules into a composite program
- linking can be done at 7 different times

  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

- (2) & (3) imply re-translation
- (4) implies a linkage editor
- (5) implies a linking loader
- (6) implies to keep resident large number of subprograms
- (7) implies dynamic linking (virtual memory)

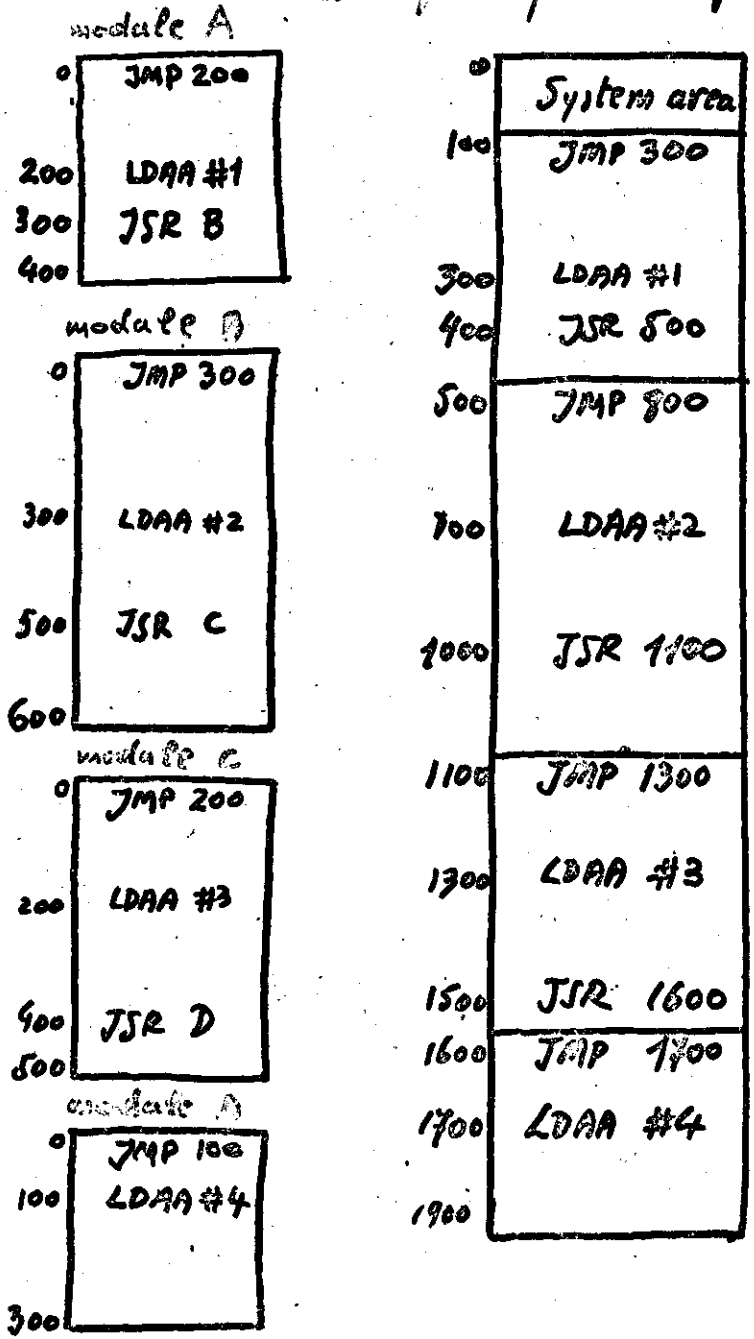


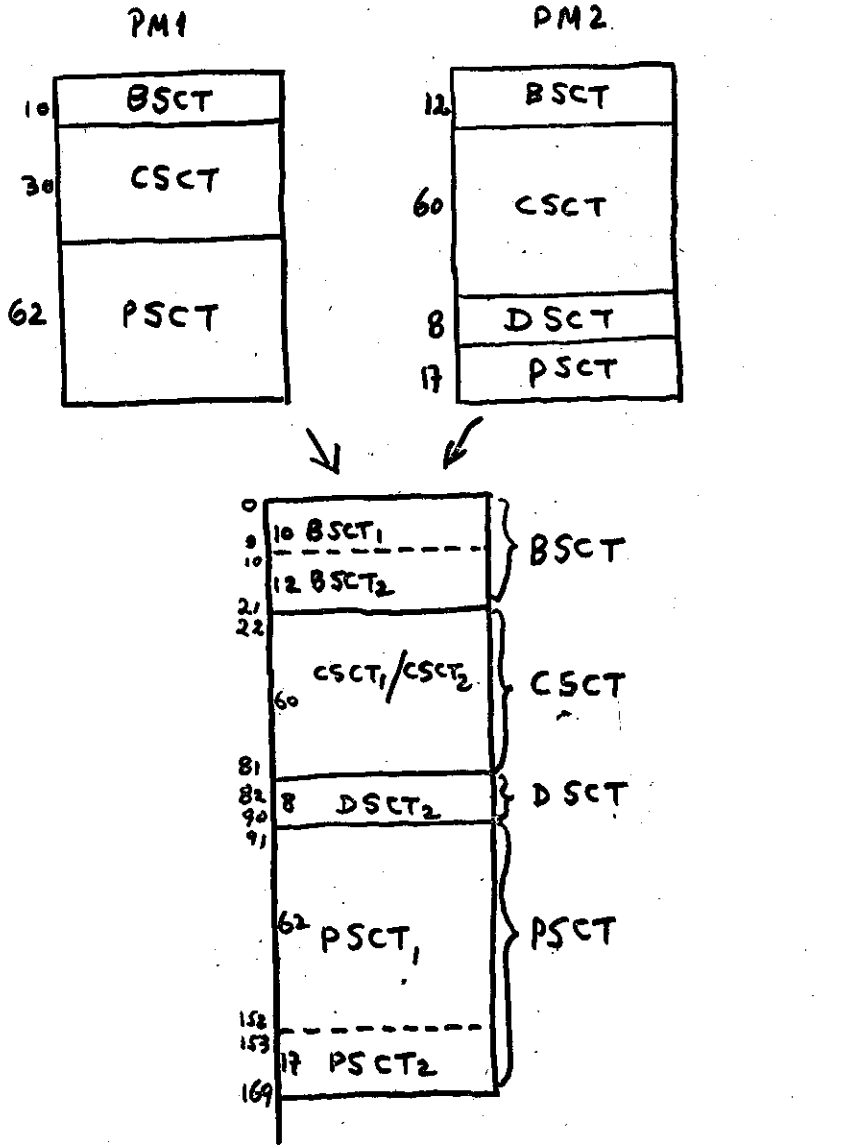
The linking process may be viewed as binding independent logical spaces into one composite logical space



Linking after translation but before execution

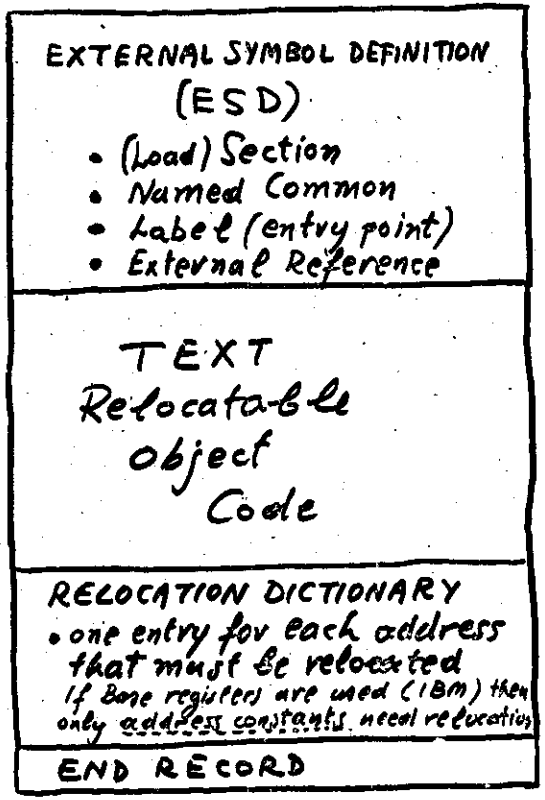
Example of Linking



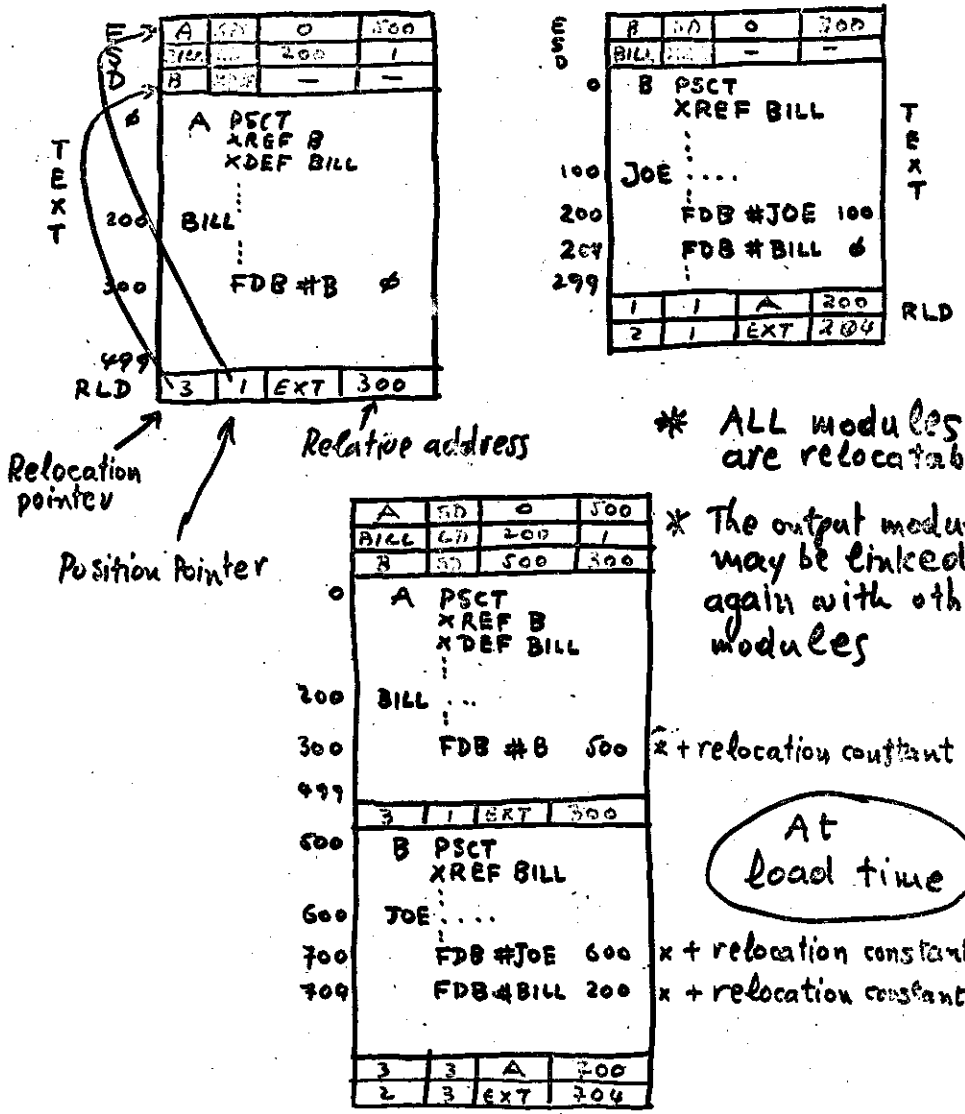


Example of load map for two modules

FORMAT OF RELOCATABLE OBJECT MODULE



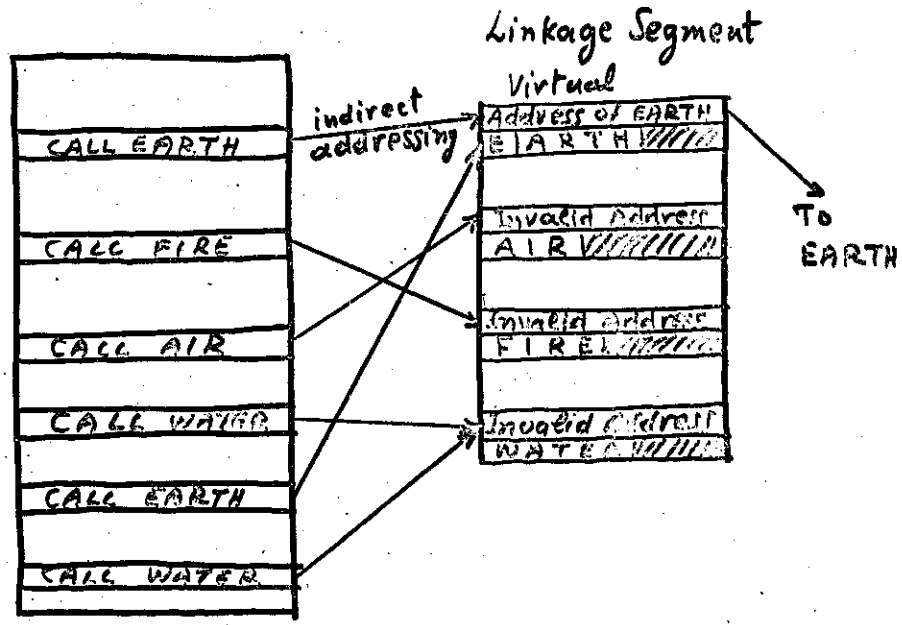
External Symbols
- External names
  - Section names
  - Entry points XDEF
- External references XREF



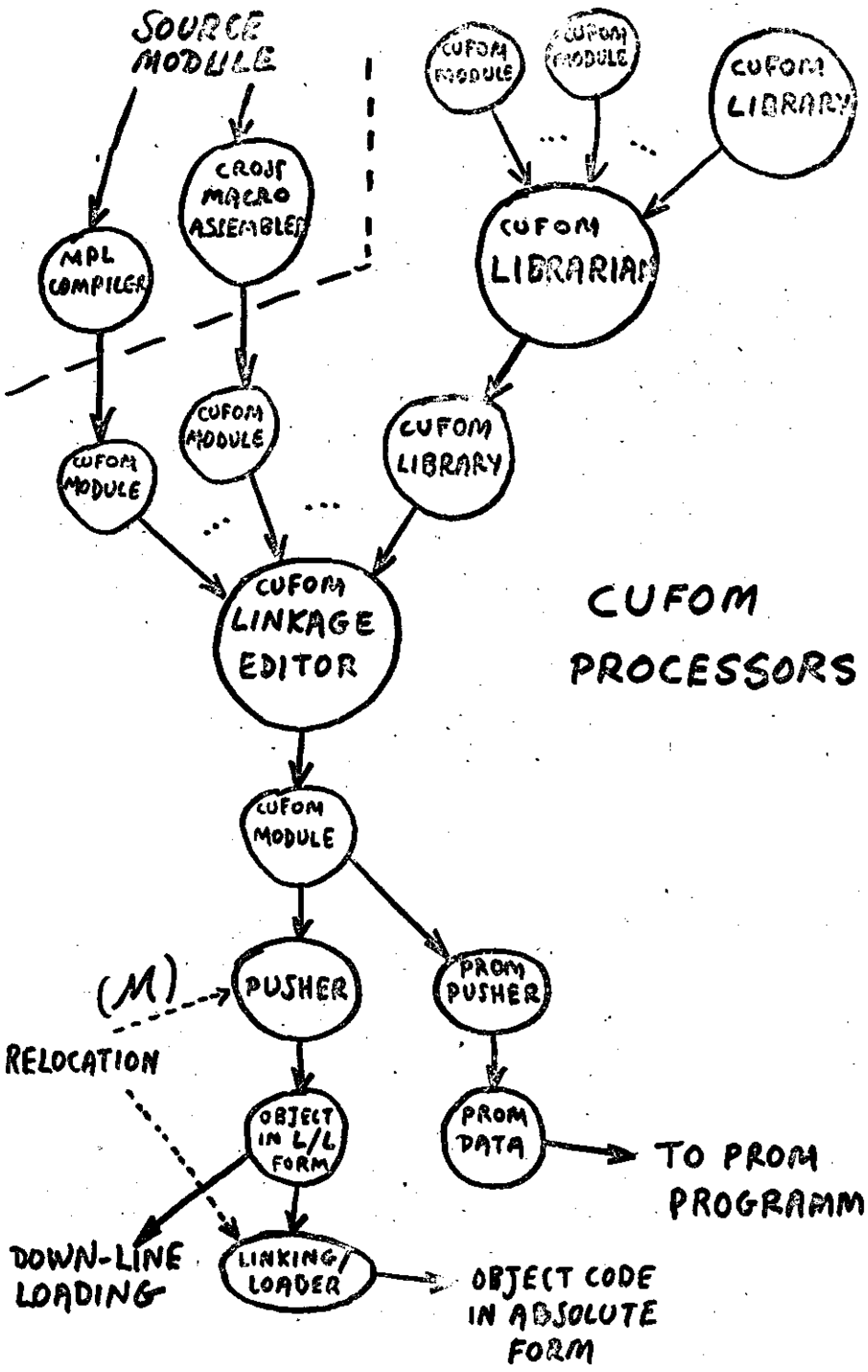
Linking example (base registers are assumed)

DYNAMIC LINKING

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

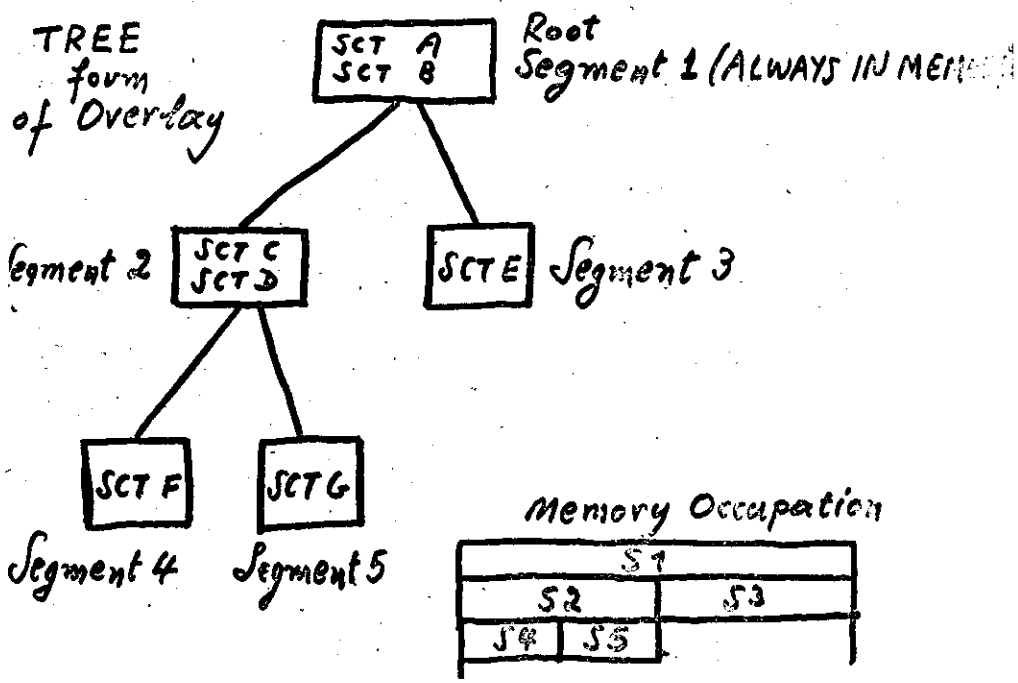


Before linking a trap occurs when first accessing Invalid Address



CUFOM PROCESSORS

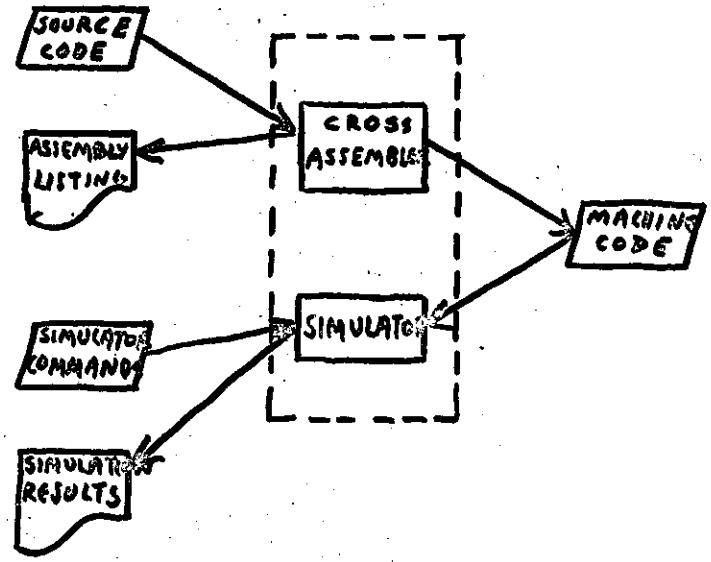
OVERLAYS



- The programmer points to the Linkage Editor, via control statements, those Sections in his program that need not reside in main memory at the same time.
- Segments that lie in a path are logically related
- When control is passed to a segment, all segments in the path between the Root and the segment are loaded in the memory if not already there.

SIMULATORS

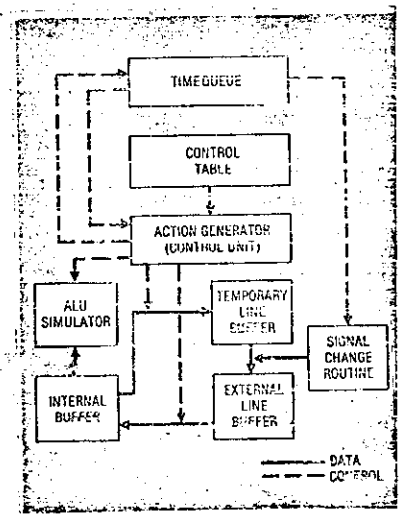
- CROSS TOOLS
- MACHINE INSTRUCTION LEVEL
- DEBUGG FACILITIES
- UNABLE TO SIMULATE TIME CRITICAL OPERATIONS
- INADEQUATE FOR A MULTI-μP SYSTEM



Operation of the M 6800 Simulator

D	Display Registers
DB n	Set Display Base to n
DF n	Set Display Flag to n
DL	Display Last Instruction
DM n,m	Display Memory (m words) start from n
EX	Exit
HR n	Set Header Count every n lines
IB n	Set Input Base to n
IM	Input Memory
LW	Last Word Address
PF	Simulate Non-Maskable Interrupts
PO	Simulate Reset Interrupt
R n	Run n Instructions
SD...	Select Display Registers
SM..	Set Memory
SR	Set Registers
T n	Trace n Instructions
TB n	Trace n Branches

M 6900 SIMULATOR COMMANDS



Chip level simulation for single chip μP