6.3 ASSEMBLERS & ASSEMBLY LANGUAGE

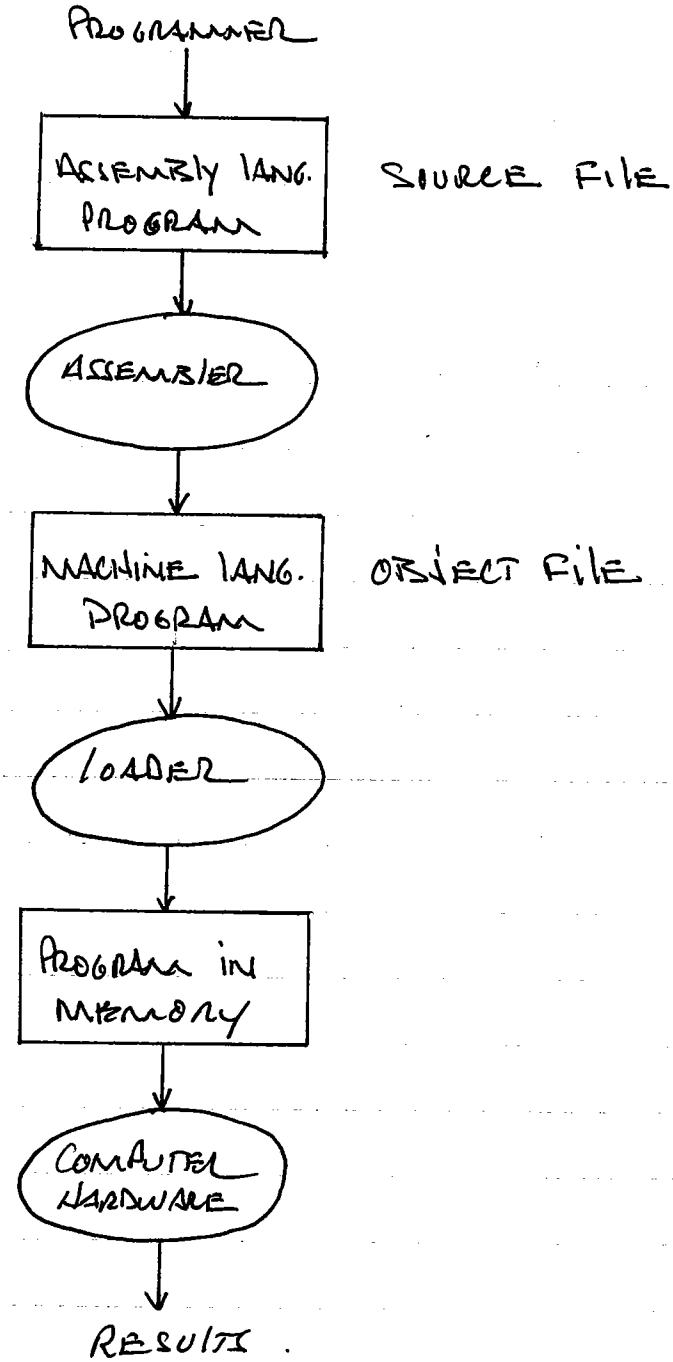
IN THE EARLY DAYS OF COMPUTING THE PROGRAMMER WOULD LOAD HIS MACHINE LANGUAGE PROGRAM BY ACTUALLY REWIRING THE COMPUTER, OR REARRANGE PLUGS ON A PLUG BOARD. THIS PROCESS BECAME AUTOMATED WITH THE ADVENT OF PUNCHED CARDS. THE PROGRAMMER WOULD PLACE HIS STACK OF CARDS, ONE FOR EACH INSTRUCTION, IN A HOPPER, PUSH A BUTTON, AND THE PROGRAM WOULD LOAD.

PROGRAMMERS WOULD NATURALLY USE SHORTHAND NOTATION WHEN WRITING PROGRAMS, THEN TRANSLATE OR ASSEMBLE THE SHORTHAND INTO MACHINE LANGUAGE. THIS SHORTHAND WAS EVENTUALLY FORMALIZED INTO ASSEMBLY LANGUAGE

AS COMPUTER TIME BECAME CHEAPER AND PROGRAMMER TIME BECAME MORE EXPENSIVE THIS TRANSLATION PROCESS WAS ALSO AUTOMATED. A PROGRAM WHICH TRANSLATES FROM ASSEMBLY LANGUAGE TO MACHINE LANGUAGE IS CALLED AN ASSEMBLER.

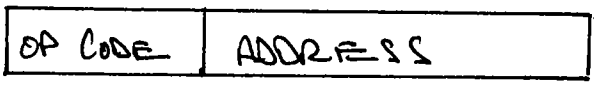
TODAY WE THINK OF ASSEMBLY LANGUAGE AS A VERY LOW LEVEL LANGUAGE. (A TRANSLATOR FOR A HIGH LEVEL LANGUAGE IS CALLED A COMPILER.)

A FLOW CHART FOR THIS PROCESS LOOKS LIKE:



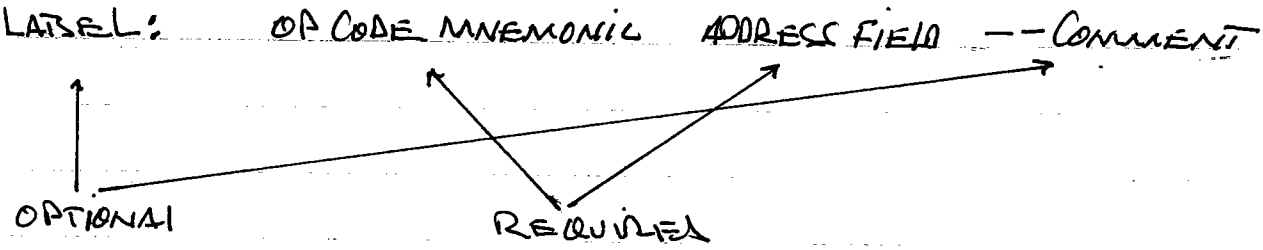
WE WILL CONSIDER A HYPOTHETICAL PROCESSOR WHOSE MACHINE LANGUAGE INSTRUCTIONS CONSIST OF A 4 BIT OP CODE, AND

A SINGLE 12 BIT ADDRESS FIELD.



THIS PROCESSOR SUPPORTS $2^4 = 16$ OPERATIONS. ITS INSTRUCTION SET IS LISTED IN FIG. 6.5 (P. 244). THE MAIN MEMORY UNIT CONTAINS 2^{12} BYTES = 4 KB.

THE ASSEMBLY LANGUAGE FOR OUR HYPOTHETICAL PROCESSOR HAS THE FOLLOWING FORMAT:



THE <u>OP CODE MNEMONIC</u> IS A SYMBOLIC NAME GIVEN TO EACH OP CODE. e.g. LOAD, STORE, CLEAR RATHER THAN 0000 0001, 0010.

OUR PROCESSOR HAS A SINGLE ALU REGISTER CALLED SIMPLY R. SOME INSTRUCTIONS MODIFY THE CONTENTS OF R, SUCH AS

LOAD X

WHICH MEANS PLACE CON(X) INTO REGISTER R (SEE FIG. 6.5)

AN IMPORTANT FEATURE OF ASSEMBLY LANGUAGES IS THAT THEY LET THE PROGRAMMER USE SYMBOLIC ADDRESSES AS WELL AS NUMERIC ADDRESSES. THUS WE CAN ATTACH AN OPTIONAL LABEL TO ANY INSTRUCTION OR PIECE OF DATA. THE LABEL IS THEN USED TO IDENTIFY THE INSTRUCTION OR DATA.

EX. BEGIN: LOAD X

⋮

JUMP BEGIN

LABELS ADD PROGRAM CLARITY AND AID IN MAINTAINABILITY

EX.

⋮

JUMP LOOP

⋮

LOOP: LOAD X

IF WE ADD AN ISTRUCTION BETWEEN THE JUMP AND LOAD INSTRUCTIONS, WE NEED NOT CHANGE THE ADDRESS FIELD IN THE JUMP INSTRUCTION.

ANOTHER FUNCTION OF ASSEMBLY LANGUAGE IS DATA GENERATION. WE SAW HOW ALGORITHMS MIGHT USE SIGNED INTEGERS, UNSIGNED INTEGERS AND FLOATING POINT REAL NUMBERS REPRESENTED IN BINARY.

A MACHINE LANGUAGE PROGRAMMER WOULD HAVE TO DO THESE CONVERSIONS BY HAND. IN ASSEMBLY LANGUAGE THE PROGRAMMER CAN HAVE THE ASSEMBLER DO THEM.

WE USE A SPECIAL ASSEMBLY LANGUAGE OP CODE CALLED A PSEUDO-OP FOR THIS. OUR SIMULATED ASSEMBLY LANGUAGE USES A SINGLE SUCH PSEUDO-OP CALLED .DATA

EX. X: .DATA -6

THIS INSTRUCTION CONVERTS -6 TO THE APPROPRIATE SIGNED INTEGER REPRESENTATION AND STORES THE VALUE IN A MEMEORY CELL WITH SYMBOLIC LABEL X. WE MAY REFER TO -6 BY THE LABEL X.

Ex. LOAD X

WHEN GENERATING DATA VALUES WE MUST BE CAREFUL NOT TO PLACE THEM IN MEMORY LOCATIONS WHERE THEY COULD BE TREATED AS INSTRUCTIONS.

WHAT HAPPENS IF WE ATTEMPT TO EXECUTE

X: .DATA -6

THE 16 BIT SIGNED INTEGER REPRESENTATION OF -6 IS.

OP CODE	ADDRESS FIELD
1 0 0 0	0 0 0 0 0 0 0 0 1 1 0

INTERPRETING WHAT SHOULD BE A DATA VALUE AS AN INSTRUCTION GIVES

JUMP 6

i.e. TAKE THE NEXT INSTRUCTION FROM CELL 6, CLEARLY NOT WAS INTENDED.

TO GUARD AGAINST THIS WE PLACE ALL .DATA COMMANDS IN A SECTION OF THE PROGRAM WHERE THEY CANNOT

BE EXECUTED, AFTER THE HALT COMMAND.

EX.

```
        LOAD   X
         :
         :
        HALT
X:      .DATA  -6
```

THERE ARE TWO MORE PSEUDO-OPS IN OUR ASSEMBLY LANGUAGE CALLED .BEGIN AND .END, WHICH TELL THE ASSEMBLER WHERE TO START AND STOP THE TRANSLATION PROCESS.

EX.

```
        .BEGIN
TOP:    LOAD    A
        STORE   B
          !
        JUMP    TOP
          !
        HALT
A:      .DATA   +5
B:      .DATA   -4
          !
        .END
```