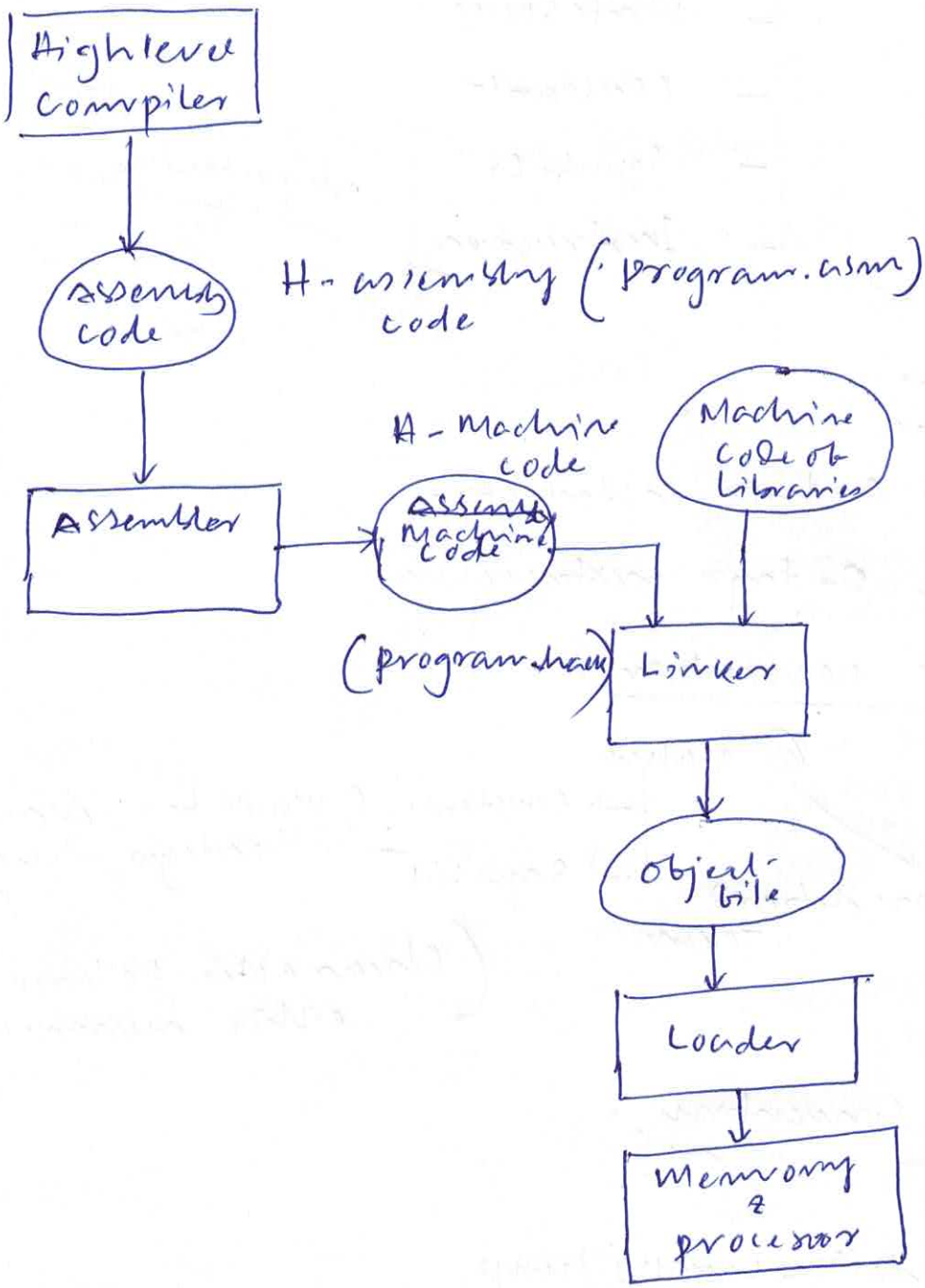
Lecture-14 [30/10/2019]

- Assembler Design -

Review of Computer System Design:-



- Assembler Design

- Takes H - assembly language as input, produces a H - machine language code.
- Requirement:
  - Syntactically correct.
  - Code generation.

(2)

Analysing the H-assembly code :-

Syntax convention :-

- Comment line
- White space
- Constant
- Symbols
- Instruction

(important)

Instructions :-

- A-type instruction
- C-type instruction

A-type convention :-

@ value

- → Constant (Must be a non-negative integer, decimal number)
- → Symbol (Character followed by either numeric or character)

predefined token

C-type convention :-

dest = comp ; jump

- → dest = comp (computer Logic)
- → comp ; jump (Jump type)

Example :-

```
(START)  @value
         M=0
         D=A
         @limit
         M=10
         @EVEN
         D;JEQ
(EVEN)   @limit
         D=A
         ~~D=D-1~~
         @2
         D=D-A
         @ENUM
         ~~M=D~~
         @EVEN
         D;JGT
(EXIT)   @EXIT
         ~~D=A~~
         JMP
```

M=M+1
M=D

Correct version :-

```
(EVEN)   @limit
         D=A
         @ENUM
(LOOP)   D=D-1
         D=D-1
         M=D
         M=M+1
         @LOOP
         D;JGT
(EXIT)   @EXIT
         0;JMP
```

parsing :-

User defined Symbol :-

START
value
limit
EVEN
ENUM
EXIT

Instruction :-

@ — A-type

M=0
D=A
D=D-A — C-type instruction

destination

Constants :-

0
10
2

- Decimal, integer
- Convert to binary (15 bit - LSB)

(4)

Dealing with symbols :-

- Labels → ROM address (Instruction Memory)
- variables → RAM address (Data Memory)
  - user defined
  - predefined variables

~~Symbol tables~~

Symbol Table :- [First pass of parsing]

Allocating Memory Location :-

Pre defined symbols :-

Symbols	RAM address
SP	0
LCL	1
ARG	2
THIS	3
THAT	4
R0-R15	0-15 (space for user defined variables)
SCREEN	16384
KBD	24576

Symbol Table Data structure

~~RAM~~ ROM

String	Integer value
START	0
EVEN	~~6~~ 7
EXIT	16

~~ROM~~ RAM

String	Integer
value	16
limit	17
ENUM	20

- Every ~~labels~~ Symbol in the symbol table needs to be unique..
- A symbol can have only one value assigned to it.

Second pass :-

- The program need to be scanned from the first instruction to the last.
- As the symbols are encountered, it will be replaced with corresponding value and the translation of instruction be completed.

Instruction Translation :-

A-type

@ value
- convert the value to binary

C-type :

dest = compj jump
- dest = comp
- compj jump

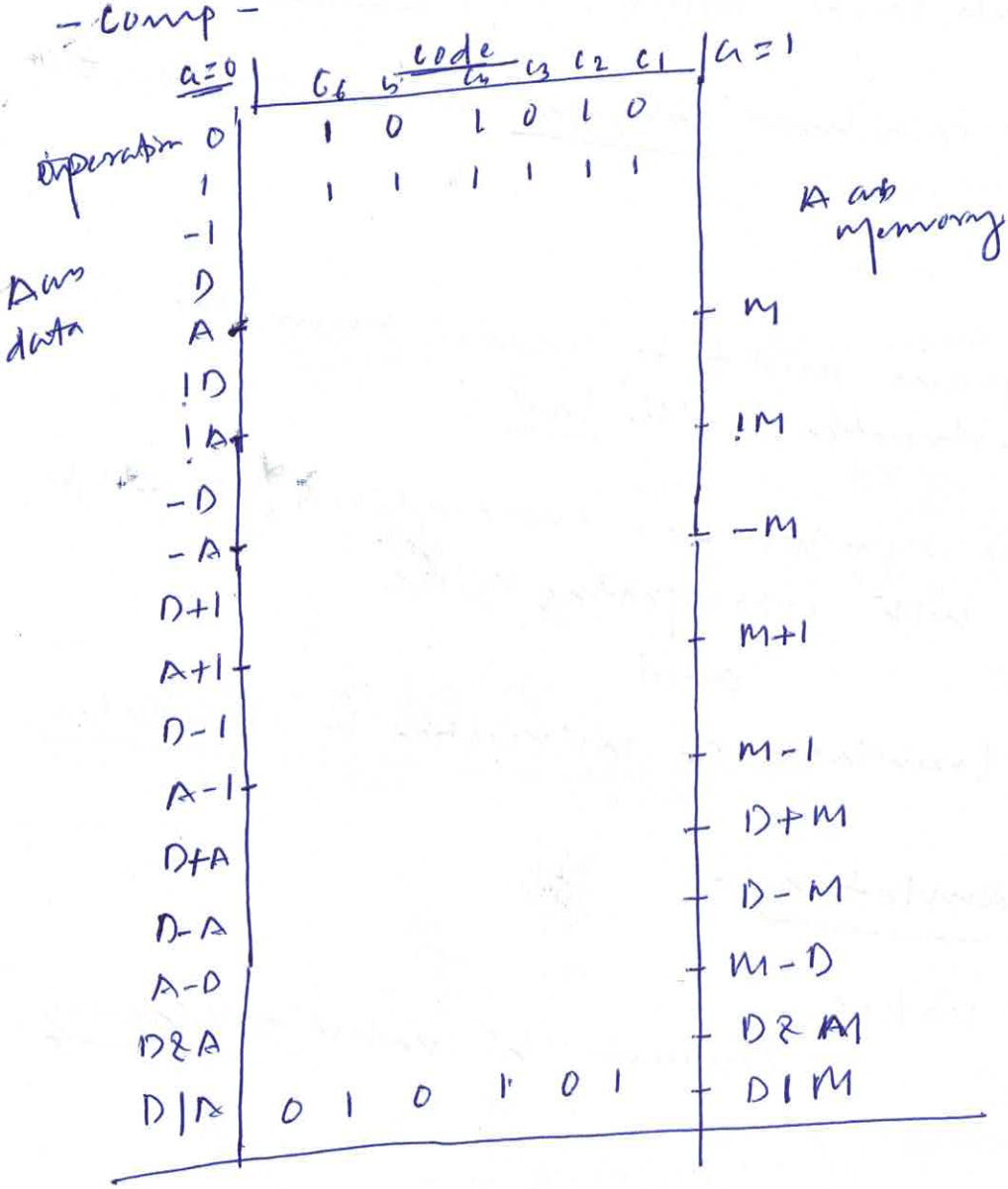
- dest -

	d_3	d_2	d_1
null	0	0	0
M	0	0	1
D	0	1	0
MD	0	1	1
A	1	0	0
AM	1	0	1
AD	1	1	0
AMD	1	1	1

- Jump -

	J_3	J_2	J_1
null	0	0	0
JGT	0	0	1
JMP	1	1	1

- Comp -


a=0
code
c_6 c_5 c_4 c_3 c_2 c_1
a=1
operation
1 0 1 0 1 0
1 1 1 1 1 1
A or memory
Acc data
0
1
-1
D
A
!D
!A
-D
-A
D+1
A+1
D-1
A-1
D+A
D-A
A-D
D&A
D|A
M
!M
-M
M+1
M-1
D+M
D-M
M-D
D&M
D|M
0 1 0 1 0 1