PA. 15. Krishnaraj Thadesar
Batch A1
Subject : SSCD

LAB. ASSIGNMENT. 1

Q.1 What errors are handled by the 2 pass assembly in pass 1 & pass 2?

→ The following errors are handled by 2 pass assembler in pass 1 & pass 2

Pass 1 errors :
- Syntax errors & incorrect instructions formats & invalid mnemonics.
- Duplicate symbols : same symbols stored more than once
- Undefined labels
- Invalid assembly directives
- Improper literal definitions

Pass 2 errors :
- Undefined symbols
- Invalid opcodes
- Address out of range
- Improper use of literals.
- Relocation errors.

Q.2 Explain SYMTAB & assembler directives EQ, START & ORIGIN.

A. SYMTAB – Symbol table

– A table maintained by the assembler to store labels / symbols & their corresponding address.
– Created in pass 1 & used in pass 2
– Entries include symbol name, address, and sometimes flags for relocation.

B. Assembler Directives:

– EQU – (Equate) Assigns a constant value to a symbol

PI EQU 3.14.

– START – Indicates starting address of program

Ex. START 200

– ORIGIN: Modifies location counter to a new specified address.

Ex. ORIGIN 200 sets location counter to 200.

Q.3 Explain LITTAB, POOLTAB & assembler directives LTORG & END.

– Literal Table –

– Store literals stored in program along with their memory addresses
– Populated during pass 1 & resolved in pass 2

– Pool Tab:

– Manage literal pools by maintaining the index of where each pool starts in LITTAB

- Created to handle multiple literals pools generated using LTORG

- Assembler directives:

1. LTORG :
   - Instructs the assembler to process all literals accumulated so far & allocate memory for them at that point
   - Ensures literals are stored close to where they are used.

2. END : Marks the end of the source program
   - Triggers the assembler to process any unassigned literals

```
START   100
MOVER   AREG, =5
        END
```

Here '=5' is processed at END if LTORG is specified.

SSCD – SEM 8

PA 15: Krishnaraj RT

LAB ASSIGNMENT 2

Q.1. Difference between single pass & pass 2 of assembler

(1) Single pass assembler:

- Process source code in one pass
- Must resolve forward references (labels used before being defined)
- Faster as it reads the source code only once
- ~~A simple~~ Requires extra memory for storing unresolved symbols for back patching

(2) 2 Pass Assembly

- Process source code in 2 Passes.
- A two pass assembler separates all symbols in pass 2 and assigns addresses in pass 2 avoiding forward reference issues.
- It is slower but ensures complete resolution of all symbols.
- It needs memory for SYMTAB, LITTAB & POOLTAB.

Q.2 Explain Variant 1 & Variant 2 with examples.

A- Variant 1 (Direct addressing mode)

- The instruction directly contains memory address of the operand.

- Simple but lacks flexibility since address is fixed

- Common in basic addressing schemes where addresses are static.

ex:

. LOAD A, 500 : Load value from memory address into A.

- 500 is the direct address where data is stored.

(2) Variant 2 (Indirect Addressing Mode)

- The instruction contains the address of memory location that stores the actual address of the operand

- Provides flexibility as memory addresses can be ~~chang~~ changed dynamically.

- Used in dynamic memory ~~al~~ allocation & relocatable programs.

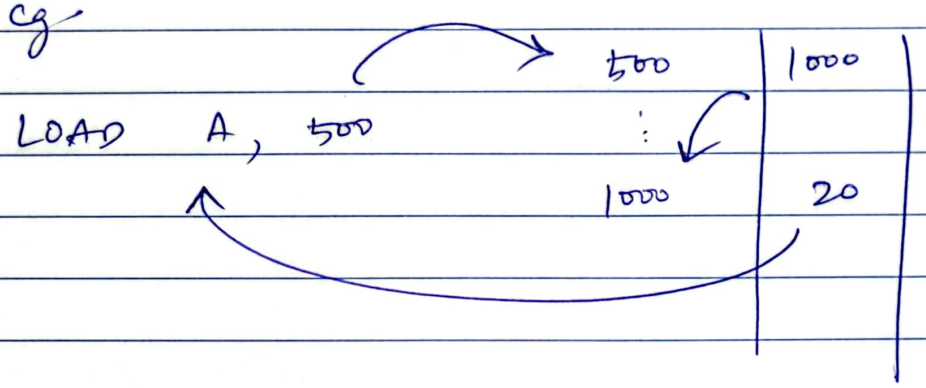
- Example

- LOAD A, (500);

- LOAD D, (3000);

If the address 500 or ~~1000~~ 3000 are the given addresses for loading ~~for~~ data, then the address stored at those locations are given to the variable

eg.



This method adds an extra step of indirection, making it more versatile than variant 1.

LAB ASSIGNMENT - 3

Q.1. What are macro instruction & defining macros?

- Macro instructions are pre-defined sequences of instructions that can be used multiple times in a program.
- They help in reducing code redundancy and simplifying complex operations.
- When a macro is invoked, the assembler expands it into its corresponding set of instructions.

→ Defining macro instructions.

- A macro is defined using a macro ~~def~~ defining block.
- Macro Name
- Parameters
- Macro Body
- Macro END Statement

Example:

```
Mymacro  MACRO   x, y
MOV  A, x
ADD  A, y
ENDM
```

Invoke:

Mymacro 5,10 ;

This expands to

MOV A,5 → ADD A
10 → MOV RESULT A

Q.2. Different types of arguments in macros:

(1) Positional Arguments.

— Arguments are passed in a fixed order.
— The assembler assigns values based on position.

(2) Keyword arguments.

— Arguments are passed using explicit names.
— Order does not matter making it more flexible

(3) Default Arguments.

— Used when no. of arguments is unknown or variable
— Typically implemented using special symbols in macro assembler.

SSCD LAB ASSIGNMENT - 9

Q.1 Explain the need of a single pass macro processor.

→ A single pass macro assembler is designed to process the source code in just one pass, meaning it expands while reading the source code without requiring a separate pre-processing step. The need for such a processor arises due to the following reasons:

(1) Faster Processing: Since the assembler does not have to make multiple passes, it speeds up macro expansion & assembly.

(2) Memory efficiency: It reduces memory usage as it avoids storing intermediate results.

(3) Immediate macro expansion: Macros are expanded as soon as they are encountered, making processing more direct & efficient.

(4) Useful for single Pass Assembler, some assemblers, especially in real-time, use a single pass approach & a macro processor is same way.

(5) Reduces Compilation Time: Since macros are expanded in the same pass as code translation the overall compilation time is reduced.

Q.2. Explain nested macro with an example.

— A nested macro is a macro that calls another macro with its definition. This allows to be modular & reusable, making assembly code more efficient & easy to manage.

eg.
```
OUTER_MACRO    MACRO    A, B
               MOV      AL, A
               INNER_MACRO  B
               ENDM

INNER_MACRO    MACRO    X
               ADD      AL, X
               ENDM

; Usage
OUTER_MACRO    5, 10
```

→ Advantages:

1. Reusability
2. Code simplification
3. Modularity.