README

compile:

    Open a command prompt and navigate to project directory.

    gcc emu.c

run:

    a in.asm or a intermediate_file.txt

>> Where assembly code or object code is stored in file

    Enter 1 for assembly code or 2 for object code

>> Set or check contents of memory locations

    To check memory location 2005h

    >>2005

    To set memory location 2005h to XYh

    >>2005 XY

    >>e to exit and proceed to assembling code and simulating

>> line by line conversion

>> Check contents of memory locations

    To check memory location 2005h

    >>2005

    To check 10 locations from 2005h

    >>2005 10

    To check register A

    >>A

    To check all registers

    >>reg

    To exit

    >>exit

FLOW CHART

```
input.asm  →  [Pass 1]  →  generate MDT, MNT
                 ↓
           [read next line]  ←─────────────────┐
                 ↓                              │
            < no line >  ──No──→  [get 1st      │
                 │                 token (t1)]  │
               Yes                    ↓         │
                 ↓                 [get 2nd      │
              [END]                 token]      │
                                      ↓         │
   [MNT[hash(t1)]   ←─Yes─  < 2nd token== >  ──No──┘
    =MDT_idx]                "macro"
        │                       │
        ↓                       │
   [read next line] → [get token → < t2== > ──→ [MDT[MDT_idx++]
        ↑              (t2)]       "mend"        ='$' (sentinel)]
        │                           │
        │                           ↓ No
        │                  [MDT[MDT_idx]
        └──────────────────  =curr_line
                             MDT_idx++]
```
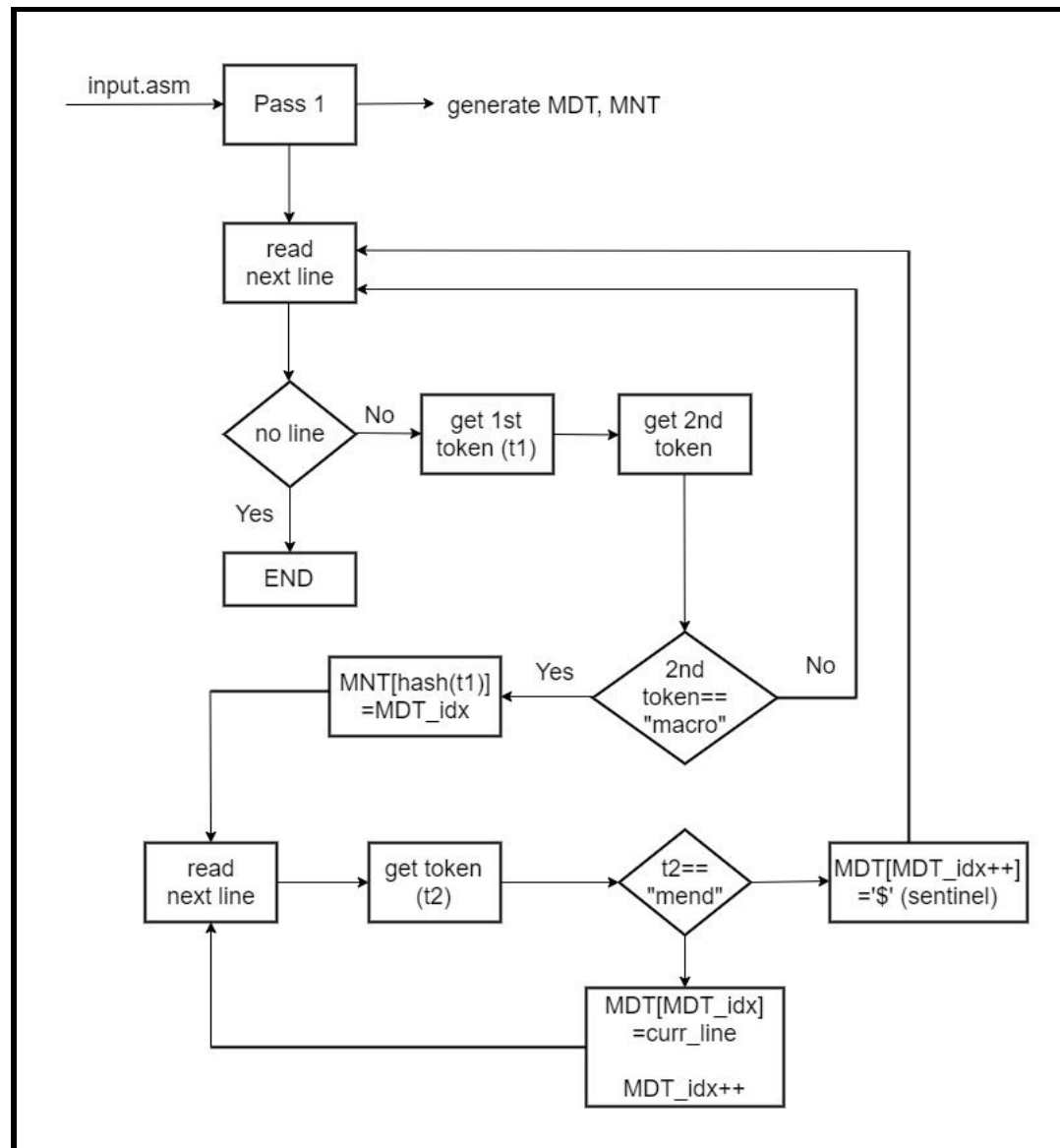
Generate macro definition table, macro name table

```
MDT, MNT ──────────────→ ┌─────────┐ ──────────→ macro_expanded.txt
                         │ Pass 2  │
input.asm ─────────────→ └────┬────┘

                              │
                              ▼
         ┌──────────┐      ◇ blank ◇    No    ◇ no ◇    No   ┌──────────┐
    ┌───→│read next │ ────→│  line  │ ──────→ │ line │ ──────→│ get 1st  │
    │    │  line    │      ◇        ◇         ◇      ◇        │token (t1)│
    │    └──────────┘           │                │           └────┬─────┘
    │                    Yes    │            Yes │                │
    │         └─────────────────┘                ▼                │
    │                                         ┌─────┐             │
    │                                         │ END │             ▼
    │                                         └─────┘
    │                                                       ◇ search ◇    ┌──────────┐
    │          ┌──────────────┐   present macro call        ◇  MNT   ◇ ──→│ get 2nd  │
    │          │  MDT_idx     │ ←──────────────────────────│ for t1  │   │token (t2)│
    │          │=MNT[hash(t1)]│                             ◇         ◇   └────┬─────┘
    │          └──────┬───────┘                                               │
    │                 │                                                       ▼
    │                 │                     present macro call          ◇ search ◇
    │          ┌──────▼───────┐   ┌──────────────────┐  after label    ◇   MNT   ◇
    │          │  MDT_idx     │ ←─│ macro_expanded[idx]│ ←─────────────│  for t2 │
    │          │=MNT[hash(t2)]│   │     +="t1"        │                ◇         ◇
    │          └──────┬───────┘   └──────────────────┘
    │                 │                (put label)
    │                 ▼
    │  Yes      ◇ MDT[MDT_idx] ◇   No   ┌──────────────────┐
    │ (macro ←─◇    =="$"      ◇ ─────→ │ macro_expanded[idx]│
    │  end)     ◇              ◇        │   +=MDT[MDT_idx]  │
    │                  ▲               │                   │
    │                  │               │      idx++        │
    │                  │               │    MDT_idx++      │
    │                  │               └─────────┬─────────┘
    │                  └─────────────────────────┘
    └────────────
```
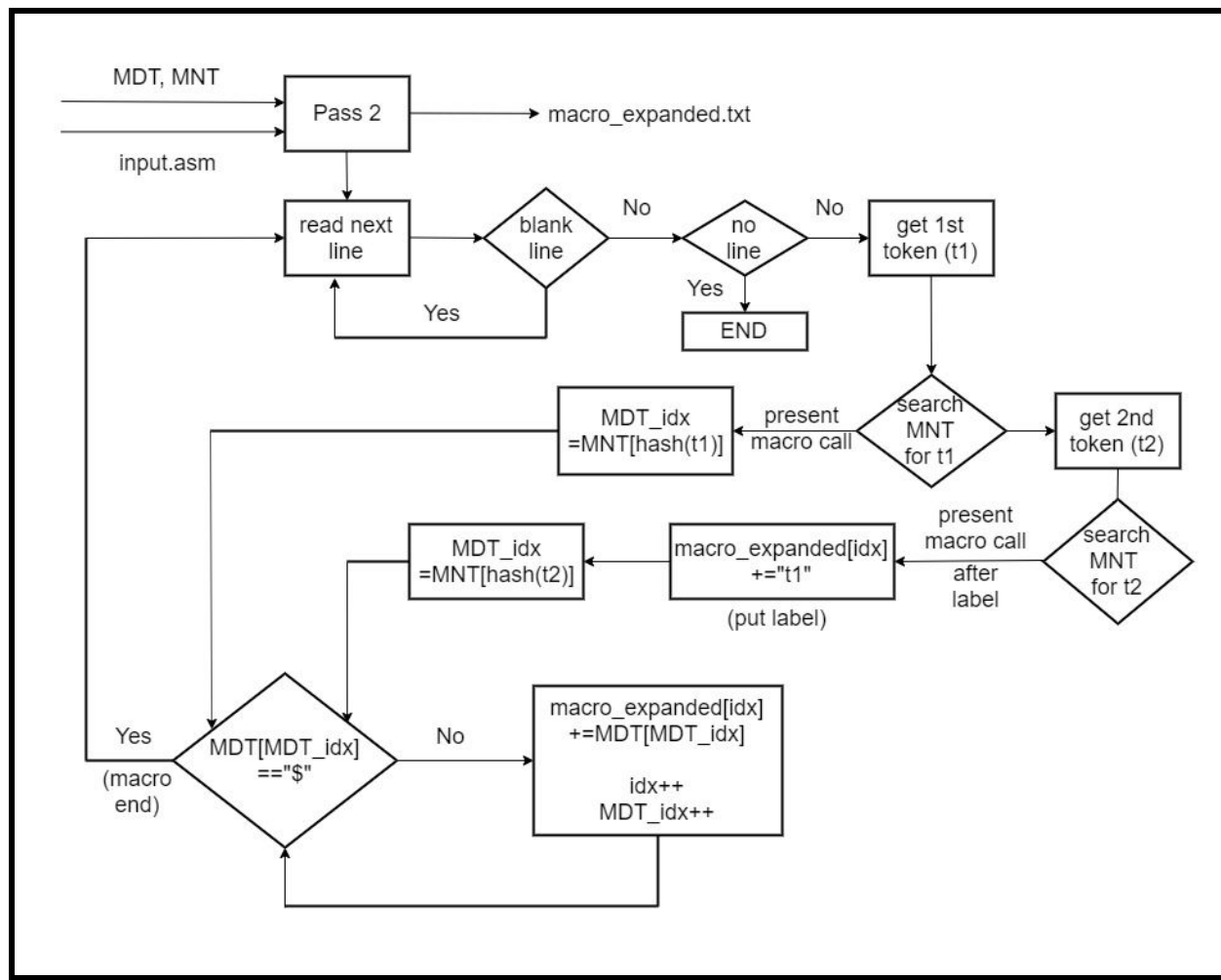
Expand macro
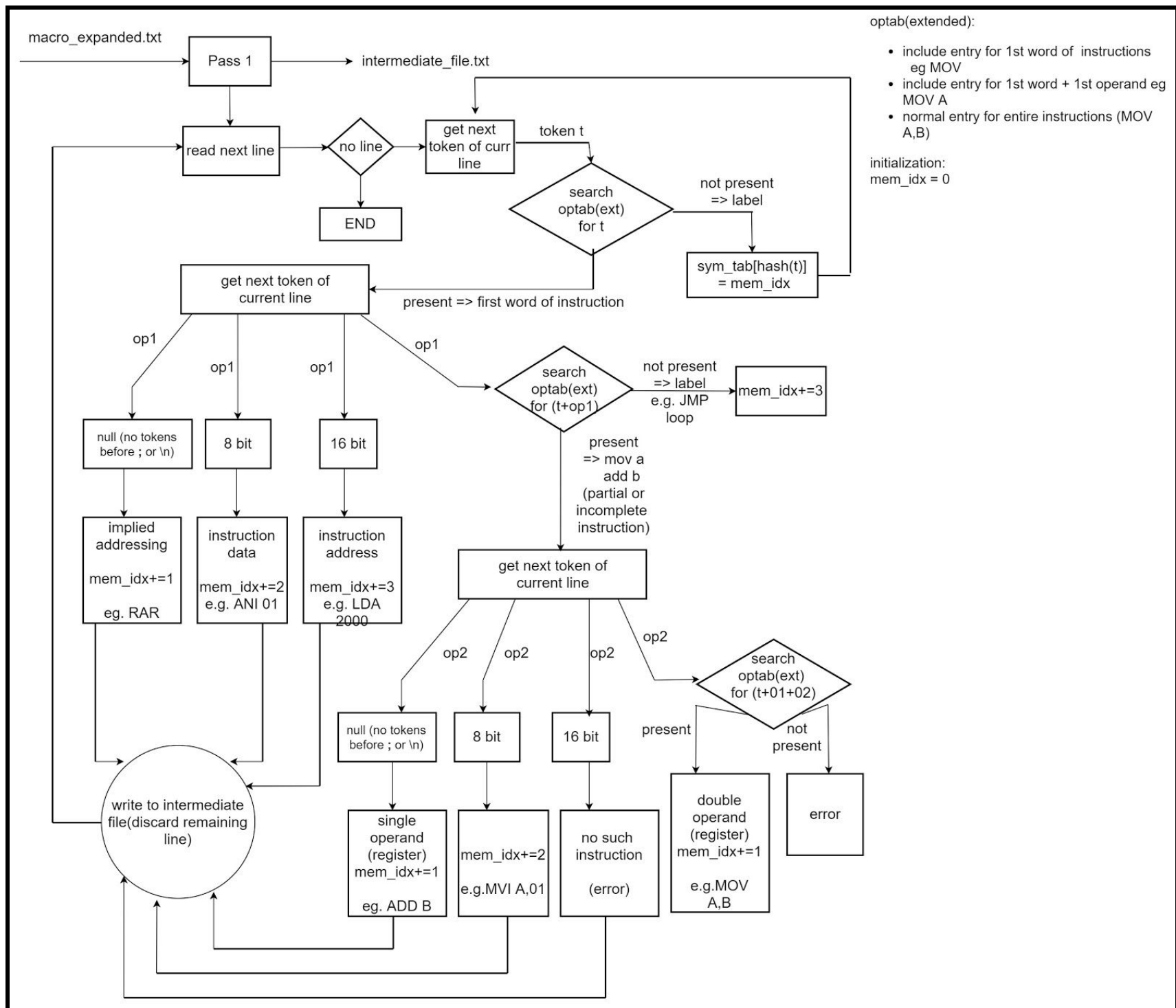
Pass 1 - Generate intermediate file

optab(extended):

- include entry for 1st word of instructions eg MOV
- include entry for 1st word + 1st operand eg MOV A
- normal entry for entire instructions (MOV A,B)

initialization:
mem_idx = 0

macro_expanded.txt

Pass 1 → intermediate_file.txt

read next line → no line → get next token of curr line → token t → search optab(ext) for t

no line → END

not present => label → sym_tab[hash(t)] = mem_idx

present => first word of instruction → get next token of current line

op1 → null (no tokens before ; or \n) → implied addressing mem_idx+=1 eg. RAR

op1 → 8 bit → instruction data mem_idx+=2 e.g. ANI 01

op1 → 16 bit → instruction address mem_idx+=3 e.g. LDA 2000

op1 → search optab(ext) for (t+op1)

not present => label e.g. JMP loop → mem_idx+=3

present => mov a add b (partial or incomplete instruction) → get next token of current line

op2 → null (no tokens before ; or \n) → single operand (register) mem_idx+=1 eg. ADD B

op2 → 8 bit → mem_idx+=2 e.g.MVI A,01

op2 → 16 bit → no such instruction (error)

op2 → search optab(ext) for (t+01+02)

present → double operand (register) mem_idx+=1 e.g.MOV A,B

not present → error

write to intermediate file(discard remaining line)

optab(extended):

- include entry for 1st word of instructions eg MOV
- include entry for 1st word + 1st operand eg MOV A
- normal entry for entire instructions (MOV A,B)

initialization :
mem_idx = 0
clear(mem)

intermediate_file.txt → **Pass 2** → object_code.txt

read next line → no line → get next token of curr line → token t

no line → END

search optab(ext) for t
- not present => label → sym_tab[hash(t)] = mem_idx
- present => first word of instruction → get next token of current line

get next token of current line:
- op1 → null (no tokens before ; or \n)
- op1 → 8 bit
- op1 → 16 bit
- op1 → search optab(ext) for (t+op1)

null (no tokens before ; or \n):
mem[mem_idx] = optab[hash(t)]
mem_idx+=1
eg. RAR

8 bit:
mem[mem_idx] = optab[hash(t)]
mem[mem_idx+1] = op1
mem_idx+=2
e.g. ANI 01

16 bit:
mem[mem_idx] = optab[hash(t)]
mem[mem_idx+1] = lower_byte(op1)
mem[mem_idx+2] = higher_byte(op1)
mem_idx+=3
e.g. LDA 2000

search optab(ext) for (t+op1):
- not present => label e.g. JMP loop → search sym_tab for op1
- present => mov a add b (partial or incomplete instruction) → get next token of current line

search sym_tab for op1:
- Present → x = sym_tab[hash(op1)]
  mem[mem_idx] = optab[hash(t)]
  mem[mem_idx+1] = lower_byte(x)
  mem[mem_idx+2] = higher_byte(x)
  mem_idx+=3
  e.g. JMP loop
- Not Present → undefined symbol

get next token of current line:
- op2 → null (no tokens before ; or \n)
- op2 → 8 bit
- op2 → 16 bit
- op2 → search optab(ext) for (t+op1+op2)

null (no tokens before ; or \n):
mem[mem_idx] = optab[hash(t+op1)]
mem_idx+=1
eg. ADD B

8 bit / 16 bit:
mem[mem_idx] = optab[hash(t)]
mem[mem_idx+1] = op1
mem_idx+=2
e.g.MVI A,01

16 bit → no such instruction (error)

search optab(ext) for (t+op1+op2):
- present → mem[mem_idx] = optab[hash(t+op1+op2)]
  mem_idx+=1
  e.g.MOV A,B
- not present → error

write from 2000h to current mem_idx of mem to object_code.txt

Pass 2 - Generate object code