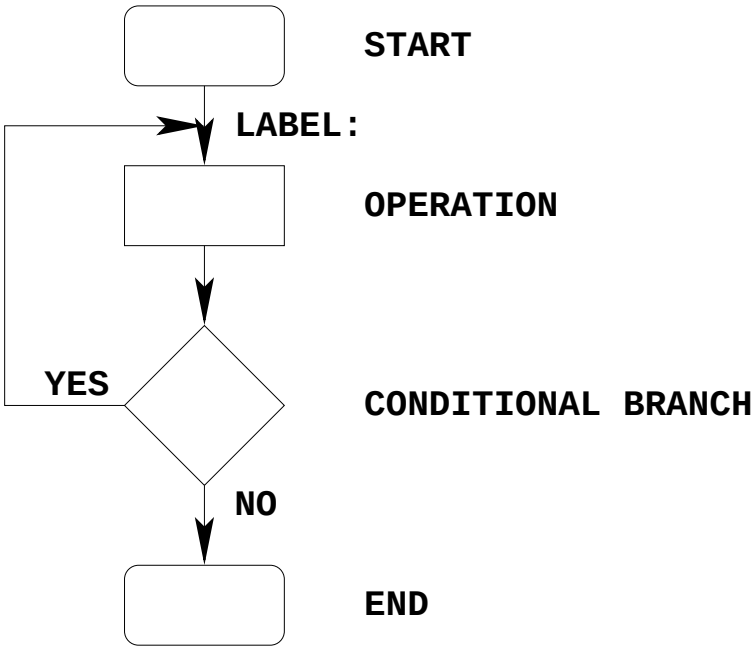
Disassembly of an HC12 Program

- It is sometimes useful to be able to convert HC12 op codes into mnemonics.
- For example, consider the hex code:

```
ADDR  DATA
---- ----------------------------------------
0800  C6  05  CE  09  00  E6  01  18  06  3F
```

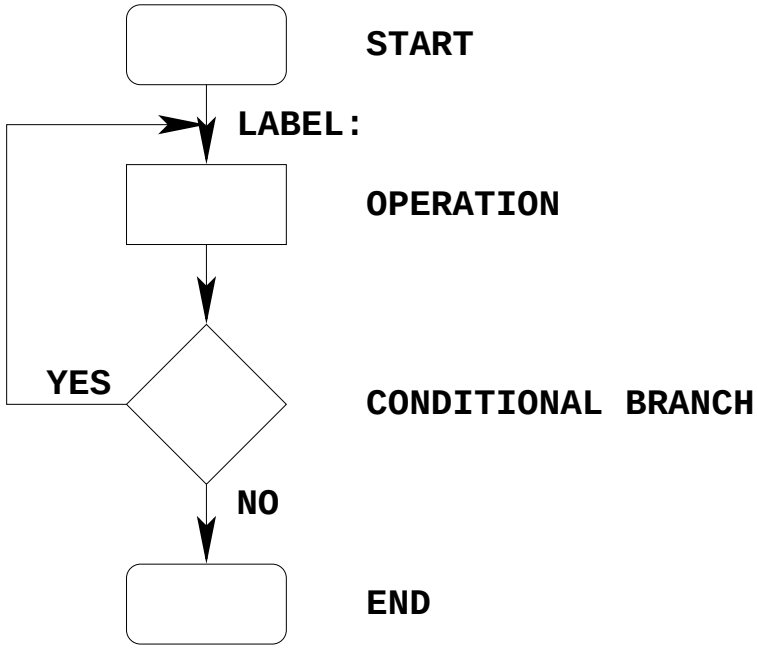
- To determine the instructions, use Table A-2 of the CPU12 Reference Manueal.
  - If the first byte of the instruction is anything other than $18, use Sheet 1 of 2 (Page A-20). From this table, determine the number of bytes of the instruction and the addressing mode. For example, $C6 is a two-byte instruction, the mnemonic is `LDAB`, and it uses the `IMM` addressing mode. Thus, the two bytes `C6 05` is the op code for the instruction `LDAB #$05`.
  - If the first byte is $18, use Sheet 2 of 2 (Page A-21), and do the same thing. For example, `18 06` is a two byte instruction, the mnemonic is `ABA`, and it uses the `INH` addressing mode, so there is no operand. Thus, the two bytes `18 06` is the op code for the instruction `ABA`.
  - Indexed addressing mode is fairly complicated to disassemble. You need to use Tables A-3 and A-5 do determine the operand.
  - For transfer (`TFR`) and exchange (`EXG`) instructions, use Table A-4 to determine which registers are being used.
- Use up all the bytes for one instruction, then go on to the next instruction.

```
C6 05     => LDAA #$05      two-byte LDAA, IMM addressing mode
CE 09 00  => LDX  #$0900    three-byte LDX, IMM addressing mode
E6 01     => LDAB 1,X       two to four-byte LDAB, IDX addressing
                            mode.  Operand 01 => 5b constant which
                            corresponds to 1,X
18 06     => ABA            two-byte ABA, INH addressing mode
3F        => SWI            one-byte SWI, INH addressing mode
```


START
LABEL:
OPERATION
YES
CONDITIONAL BRANCH
NO
END

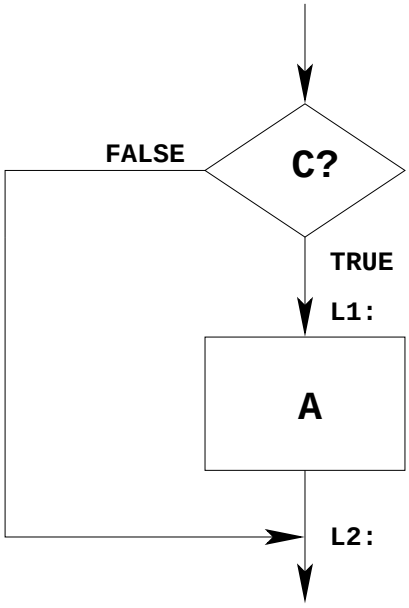
Writing Assembly Language Programs — Use Flowcharts to Help Plan Program Structure

Flow chart symbols:



IF-THEN Flow Structure

```
if (C)
{
  A;
}
```



EXAMPLE:

```
IF (A<10)
{
    var = 5;
}
```

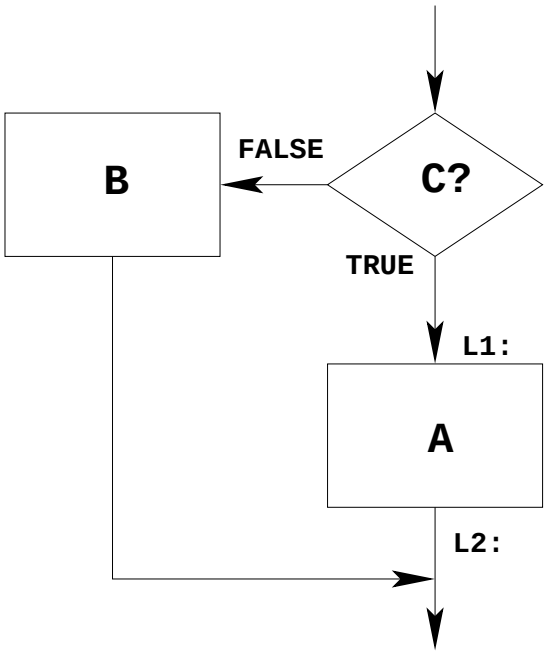
```
        CMPA    #$10
        BLT     L1
        BRA     L2
L1:     LDAB    #$5
        STAB    var
L2:     next instruction
```

OR:

```
        CMPA    #$10
        BGE     L2
        LDAB    #$5
        STAB    var
L2:     next instruction
```

IF-THEN-ELSE Flow Structure

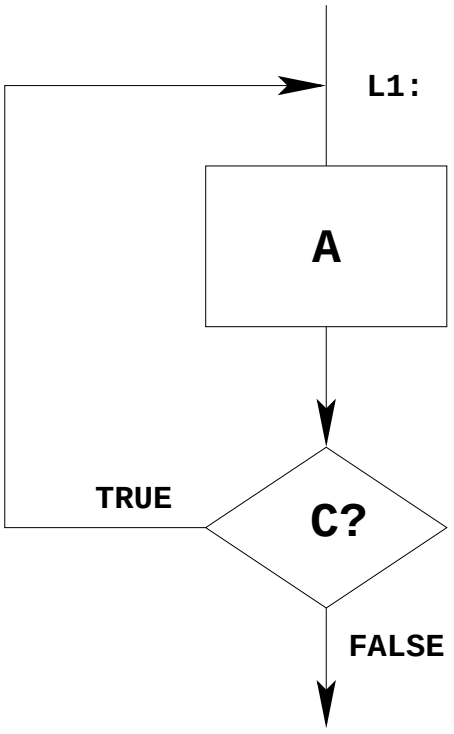
```
if (C)
{
  A;
}
else
{
  B;
}
```




```
if (A<10)                       CMPA   #$10
{                               BLT    L1
   var = 5;                     CLR    VAR
}                               BRA    L2
else                      L1:   LDAB   #$5
{                               STAB   var
   var = 0;               L2:   next instruction
}
```

DO WHILE Flow Structure

```
do
{
  A;
}
while (C);
```



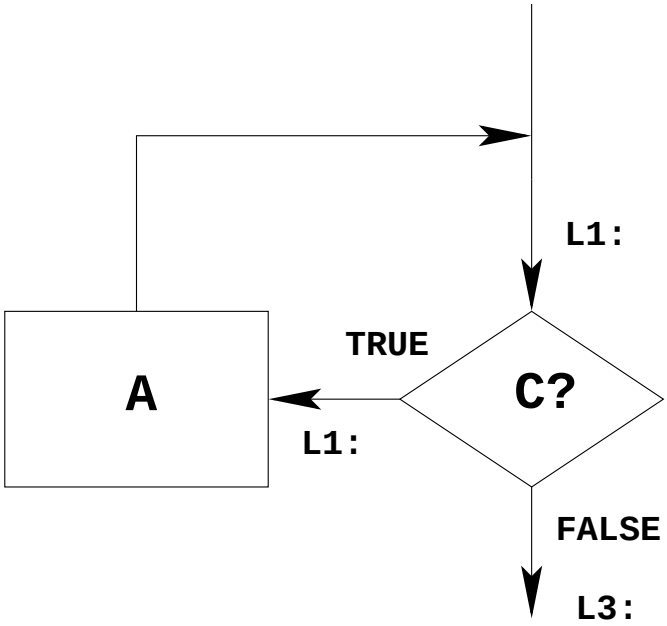
EXAMPLE:

```
i = 0;
do
{
   table[i] = table[i]/2;
   i = i+1;
}
while (i <= LEN);
```

```
      LDX    #table
      CLRA
L1:   ASR    0,X+
      INCA
      CMPA   #LEN
      BLE    L1
```

WHILE Flow Structure

```
while (C)
{
  A;
}
```



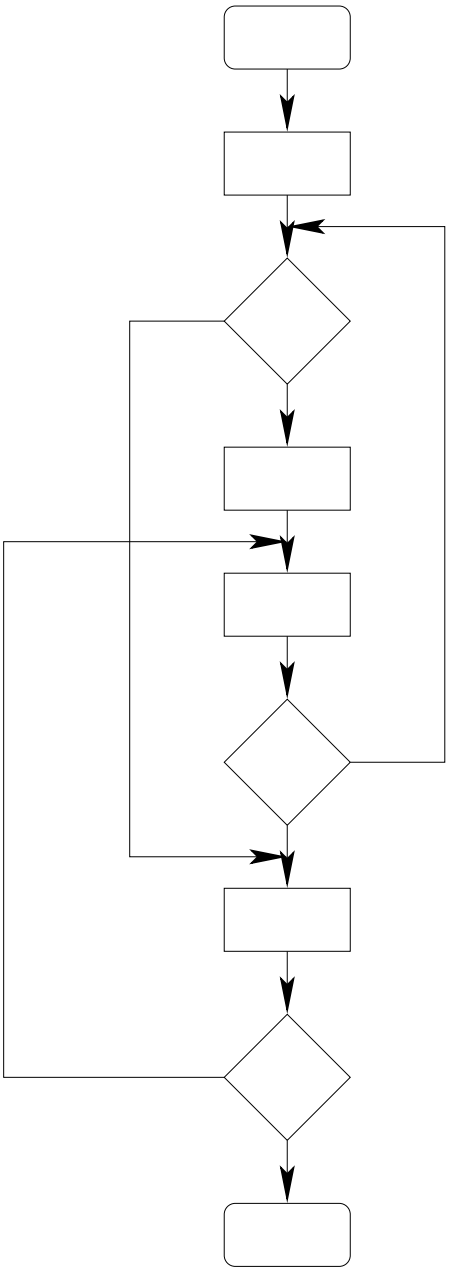
EXAMPLE:

```
i = 0;
while (i <= LEN)
{
   table[i] = table[i]*2;
   i = i + 1;
}
```

```
        LDX    #table
        CLRA
L1:     CMPA   #$LEN
        BLT    L1
        BRA    L3
L1:     ASL    0,X+
        INCA
        BRA    L1
L3:     next instruction
```

Use Good Structure When Writing Programs — Do Not Use Spaghetti Code

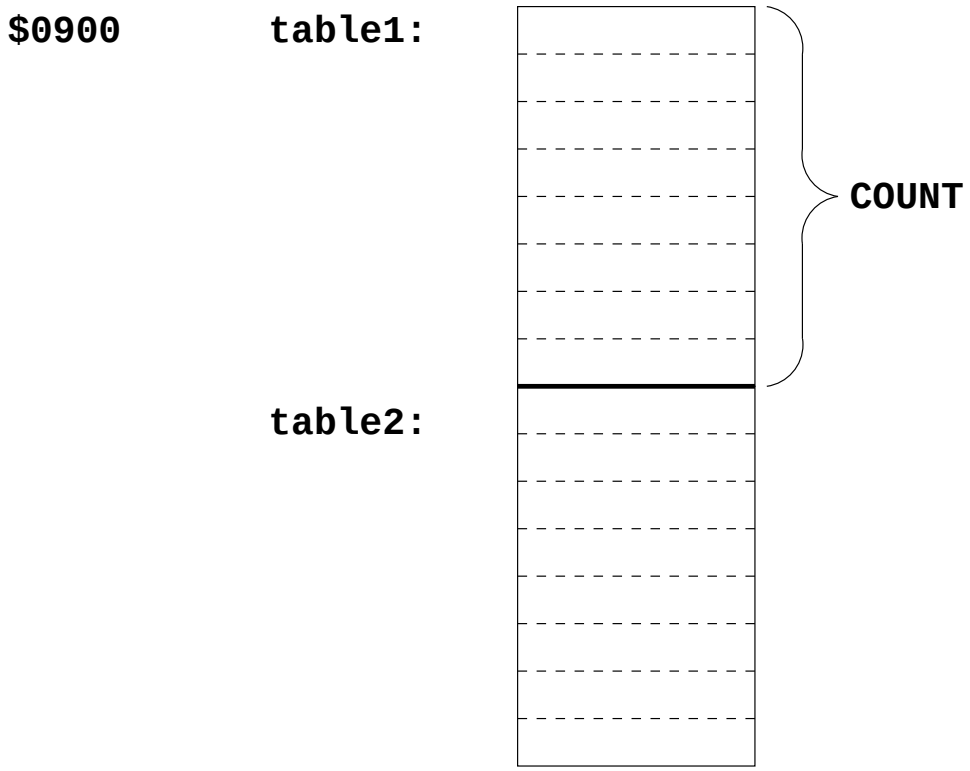
SPAGHETTI CODE
DO NOT USE



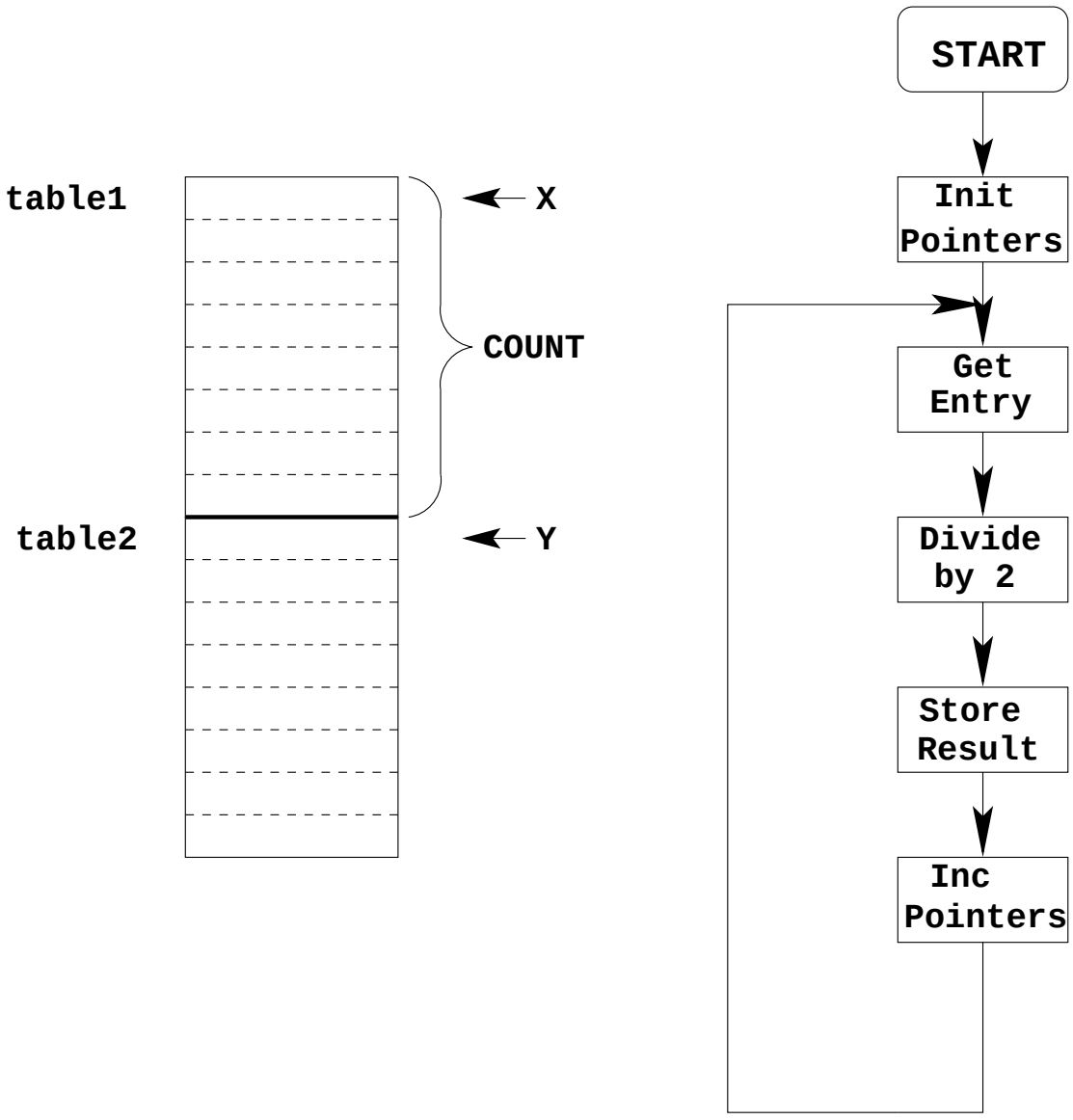
Example Program: Divide a table of data by 2

Problem: Start with a table of data. The table consists of 5 values. Each value is between 0 and 255. Create a new table whose contents are the original table divided by 2.

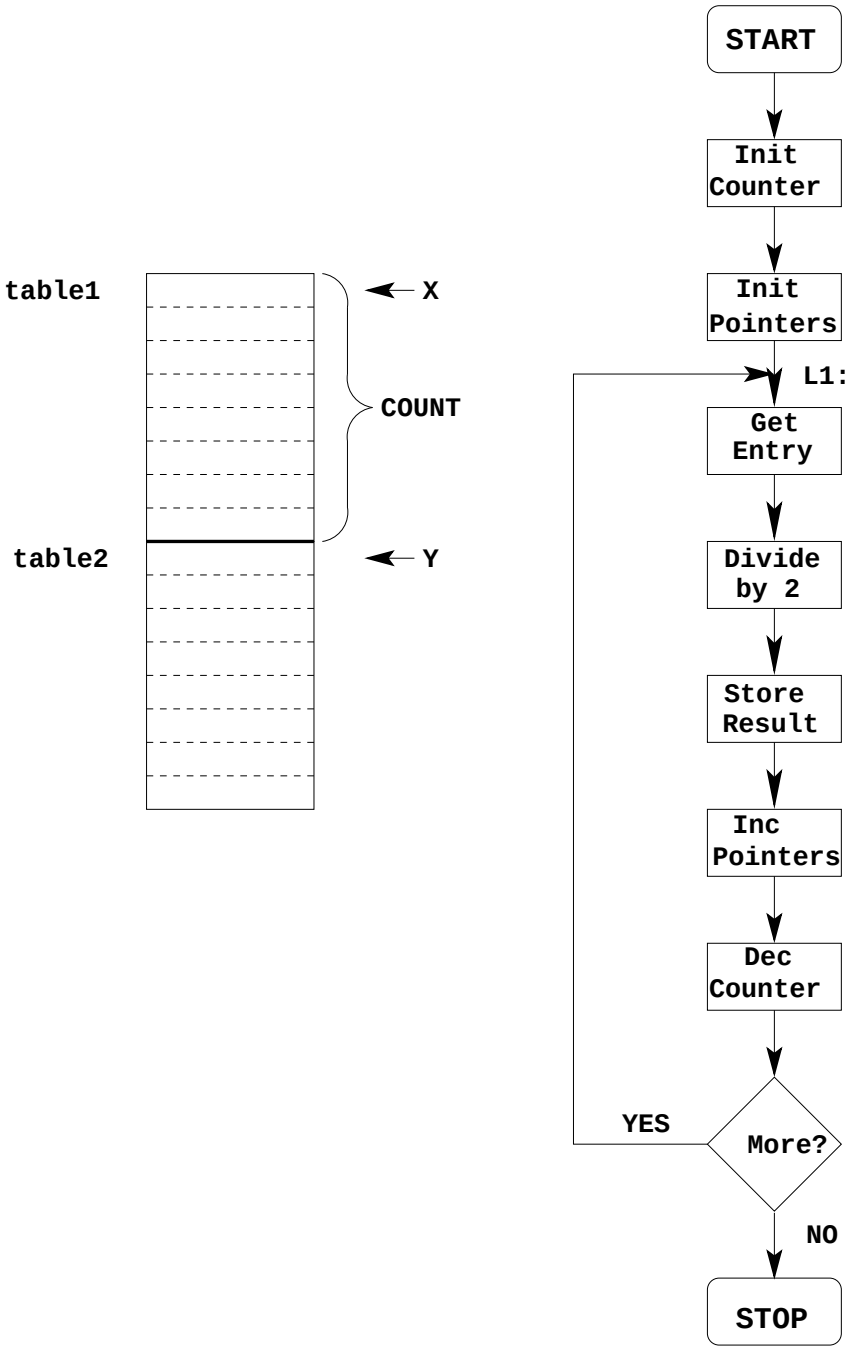
1. Determine where code and data will go in memory.
   Code at $0800, data at $0900.
2. Determine type of variables to use.
   Because data will be between 0 and 255, can use unsigned 8-bit numbers.
3. Draw a picture of the data structures in memory:



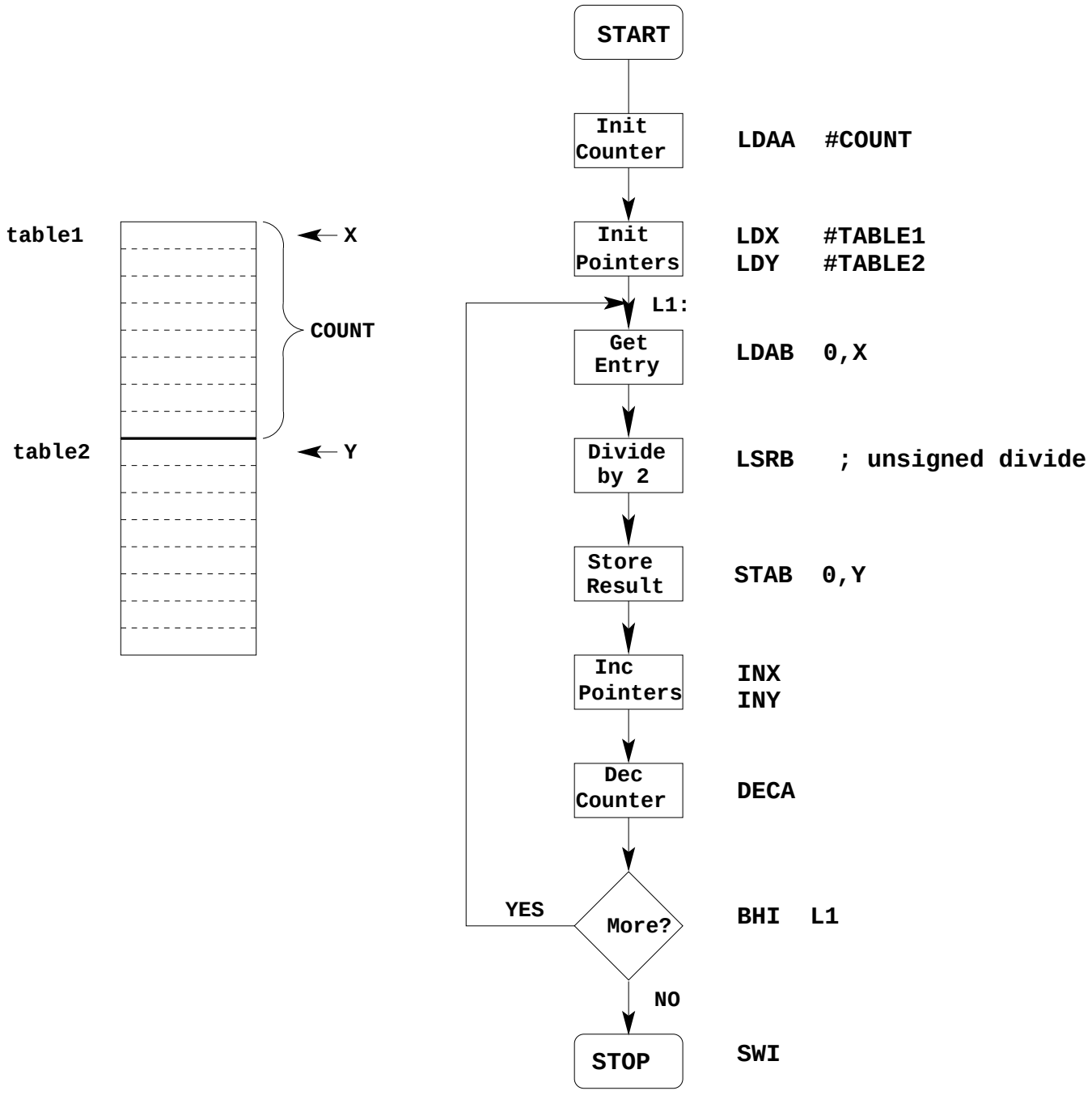
4. Strategy: Because we are using a table of data, we will need pointers to each table so we can keep track of which table element we are working on.
   Use the X and Y registers as pointers to the tables.
5. Use a simple flow chart to plan structure of program.



6. Need a way to determine when we reach the end of the table.
   One way: Use a counter (say, register A) to keep track of how many elements we have processed.



7. Add code to implement blocks:



8. Write program:

```
; Program to divide a table by two
; and store the results in memory

prog:    equ     $0800
data:    equ     $0900

count:   equ     5

CODE:    section .text    ;The stuff which follows is program code
         org     prog     ;set program counter to 0x0800
         ldaa    #count   ;Use B as counter
         ldx     #table1  ;Use X as data pointer to table1
         ldy     #table2  ;Use Y as data pointer to table2
l1:      ldab    0,x      ;Get entry from table1
         lsrb             ;Divide by two (unsigned)
         stab    0,y      ;Save in table2
         inx              ;Increment table1 pointer
         iny              ;Increment table2 pointer
         deca             ;Decrement counter
         bhi     l1       ;counter > 0 => more entries to divide
         swi              ;Done


DATA:    section .data    ;The stuff which follows is data
         org     data
table1:  dc.b    $07,$c2,$3a,$68,$F3
table2:  ds.b    count
```

9. Advanced: Optimize program to make use of instructions set efficiencies:

```
; Program to divide a table by two
; and store the results in memory

prog:    equ     $0800
data:    equ     $0900

count:   equ     5

CODE:    section .text    ;The stuff which follows is program code
         org     prog     ;set program counter to 0x0800
         ldaa    #count   ;Use B as counter
         ldx     #table1  ;Use X as data pointer to table1
         ldy     #table2  ;Use Y as data pointer to table2
l1:      ldab    0,x+     ;Get entry from table1; then inc pointer
         lsrb             ;Divide by two (unsigned)
         stab    0,y+     ;Save in table2; then inc pointer
         dbne    a,l1     ;Decrement counter; if not 0, more to do
         swi              ;Done


DATA:    section .data    ;The stuff which follows is data
         org     data
table1:  dc.b    $07,$c2,$3a,$68,$F3
table2:  ds.b    count
```

TOP-DOWN PROGRAM DESIGN

- PLAN DATA STRUCTURES IN MEMORY
- START WITH A LARGE PICTURE OF PROGRAM STRUCTURE
- WORK DOWN TO MORE DETAILED STRUCTURE
- TRANSLATE STRUCTURE INTO CODE
- OPTIMIZE FOR EFFICENCY —
  DO NOT SACRIFICE CLARITY FOR EFFICIENCY