Lecture 8

February 3, 2012

Writing Assembly Language Programs

- Use flow charts to lay out structure of program
- Use common flow structures
  - if-then
  - if-then-else
  - do-while
  - while
- Do not use spaghetti code
- Plan structure of data in memory
- Top-down Design
  - Plan overall structure of program
  - Work down to more detailed program structure
  - Implement structure with instructions
- Optimize program to make use of instruction efficiencies
- Do not sacrifice clarity for efficiency or speed

Addition of Hexadecimal Numbers

ADDITION:

C bit set when result does not fit in word

V bit set when P + P = N
N + N = P

N bit set when MSB of result is 1

Z bit set when result is 0

7A	2A	AC	AC
+52	+52	+8A	+72
CC	7C	36	1E
C: 0	C: 0	C: 1	C: 1
V: 1	V: 0	V: 1	V: 0
N: 1	N: 0	N: 0	N: 1
Z: 0	Z: 0	Z: 0	Z: 0

Subtraction of Hexadecimal Numbers

SUBTRACTION:

C bit set on borrow (when the magnitude of the subtrahend is greater than the minuend)

V bit set when N − P = P
P − N = N

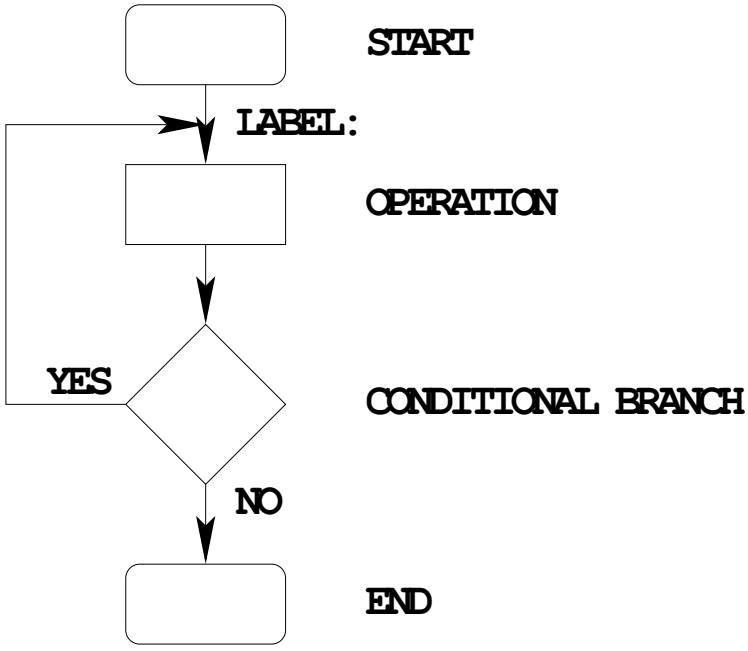
N bit set when MSB is 1

Z bit set when result is 0

7A	8A	5C	2C
−5C	−5C	−8A	−72
1E	2E	D2	BA
C: 0	C: 0	C: 1	C: 1
V: 0	V: 1	V: 1	V: 0
N: 0	N: 0	N: 1	N: 1
Z: 0	Z: 0	Z: 0	Z: 0

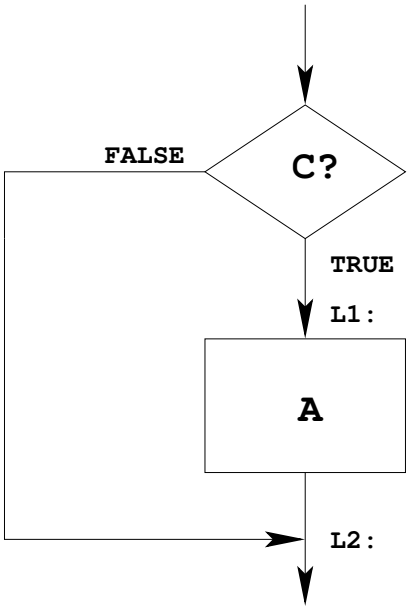
Writing Assembly Language Programs — Use Flowcharts to Help Plan Program Structure

Flow chart symbols:



IF-THEN Flow Structure

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



EXAMPLE:

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

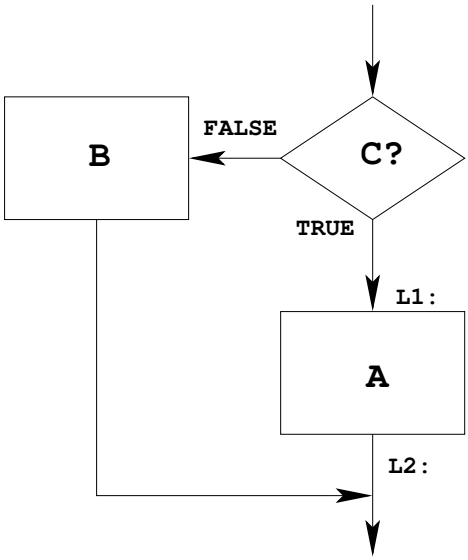
```
        CMPA    #10 ; if (A < 10)
        BLT     L1  ; signed numbers
        BRA     L2
L1:     LDAB    #5  ; var = 5;
        STAB    var
L2:     next instruction
```

OR:

```
        CMPA    #10 ; if (A < 10)
        BGE     L2  ; signed numbers
        LDAB    #5  ; var = 5
        STAB    var
L2:     next instruction
```

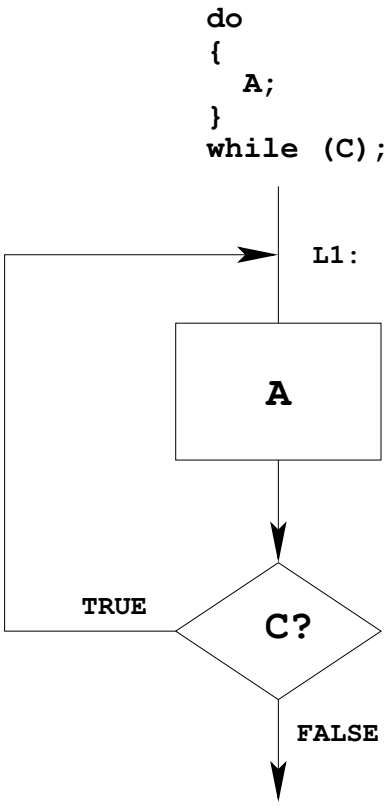
IF-THEN-ELSE Flow Structure

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




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

DO WHILE Flow Structure

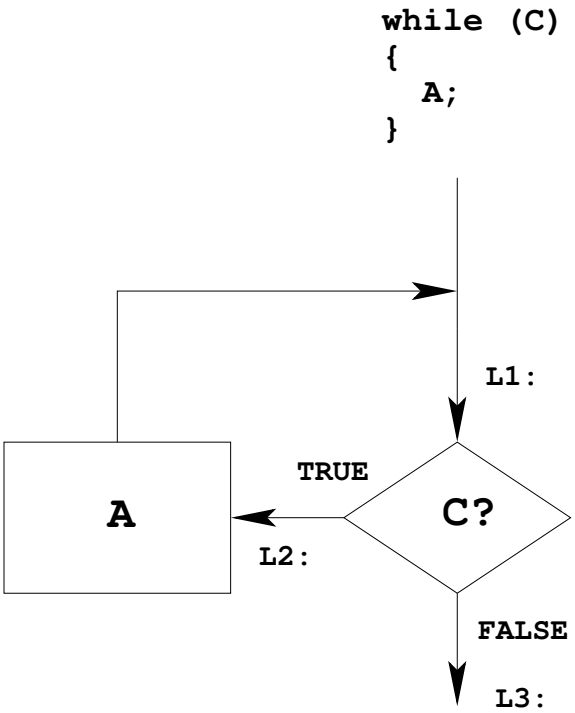


EXAMPLE:

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

```
      LDX    #table
      CLRA              ; i = 0
L1:   ASR    1,X+       ; table[i] /= 2
      INCA              ; i = i+1
      CMPA   #LEN       ; while (i <= 10)
      BLE    L1         ; unsigned numbers
```

WHILE Flow Structure

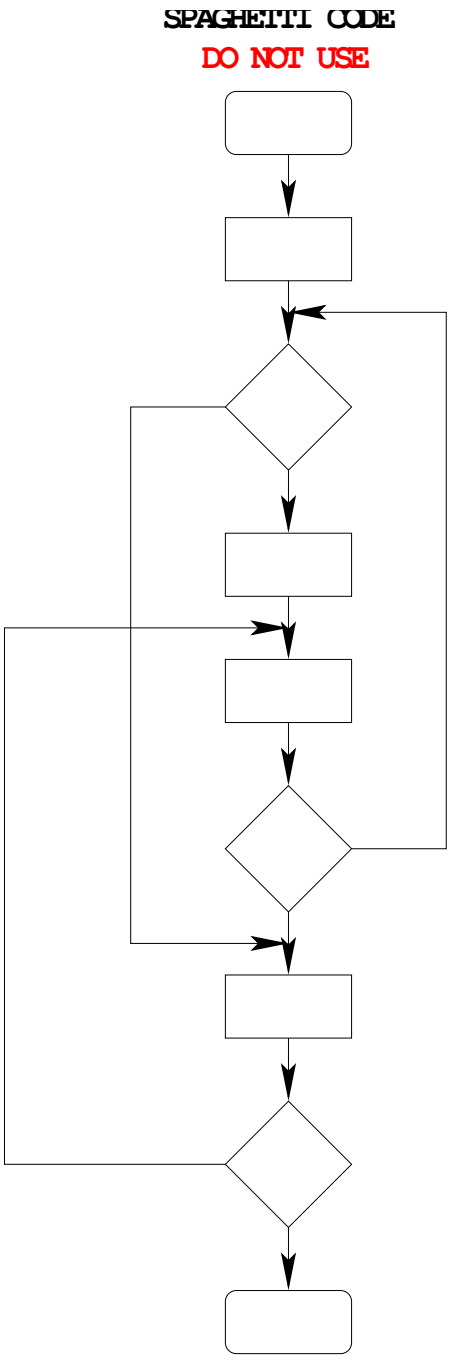


EXAMPLE:

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

```
      LDX    #table
      CLRA              ; i = 0
L1:   CMPA   #LEN       ; while (i <= LEN)
      BLT    L2
      BRA    L3
L2:   ASL    1,X+       ; table[i] /= 2
      INCA              ; i = i + 1
      BRA    L1
L3:   next instruction
```

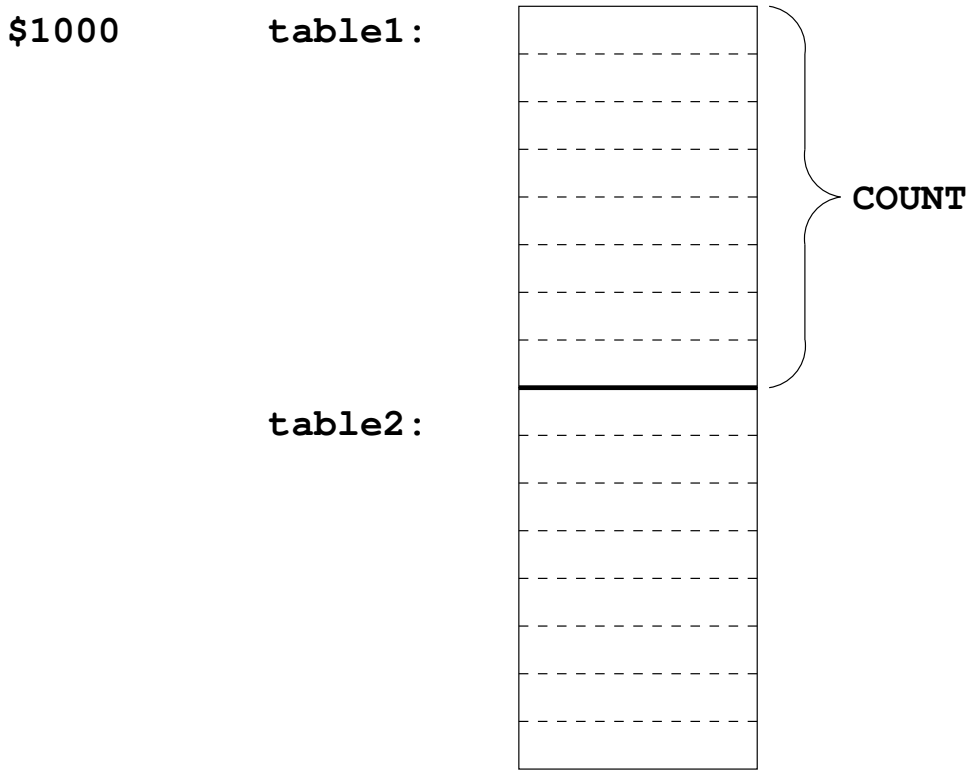
Use Good Structure When Writing Programs — Do Not Use Spaghetti Code



Example Program: Divide a table of data by 2

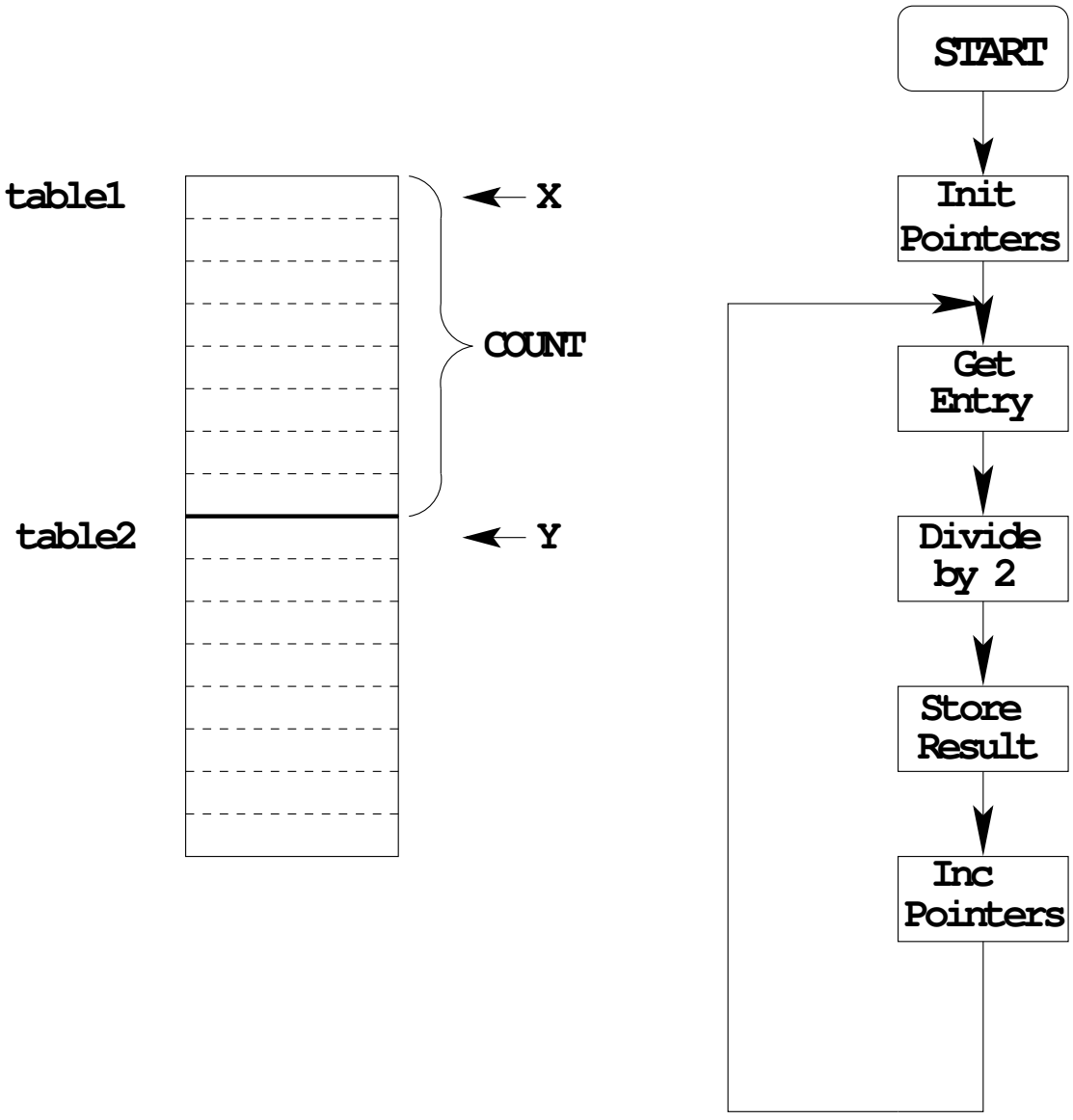
Problem: Start with a table of data. The table consists of 5 values. Each value is between 0 and 255. Create a new table whose contents are the original table divided by 2.

1. Determine where code and data will go in memory.
   Code at $2000, data at $1000.
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:

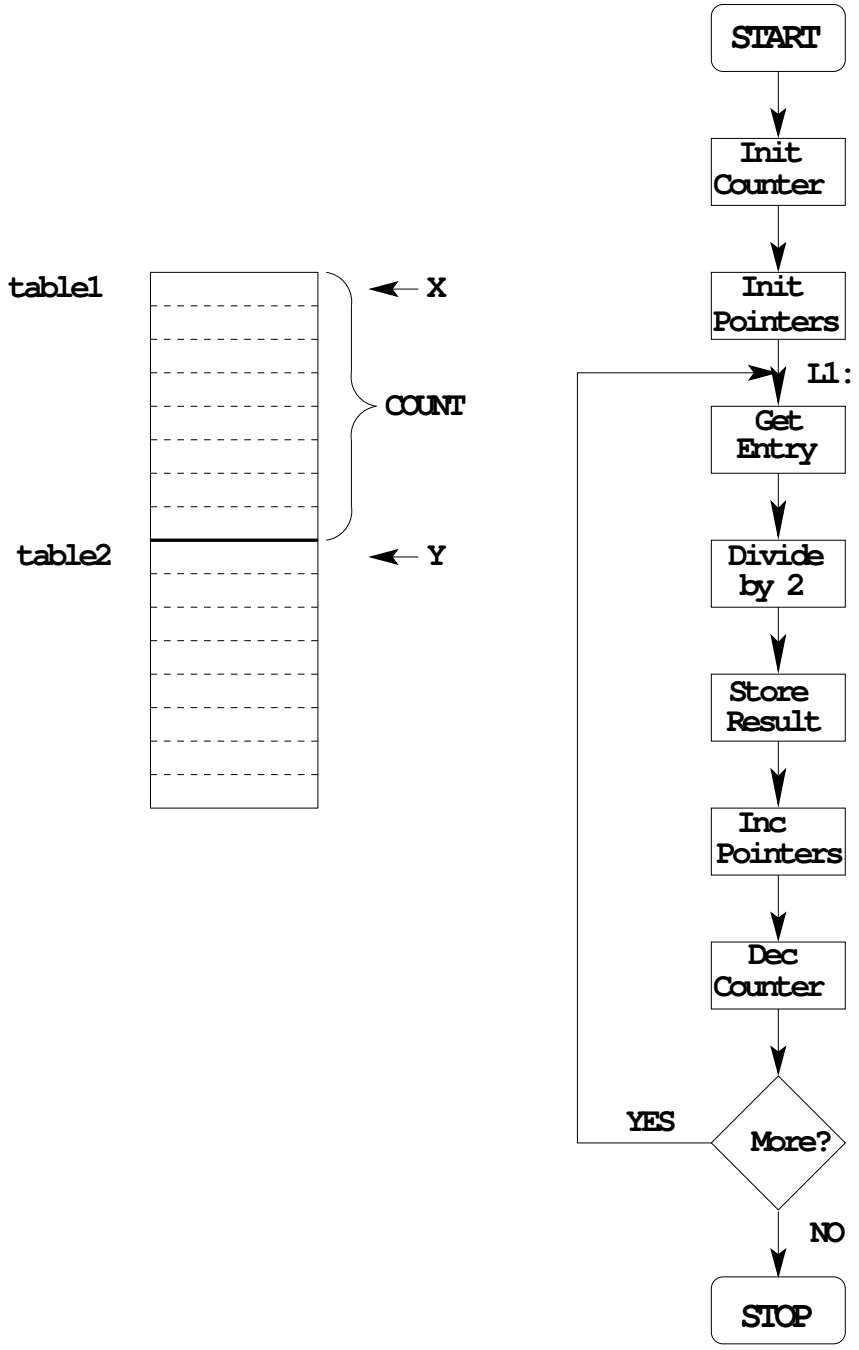


4. Strategy: Because we are using a table of data, we will need pointers to each table so we can keep track of which table element we are working on.
   Use the X and Y registers as pointers to the tables.

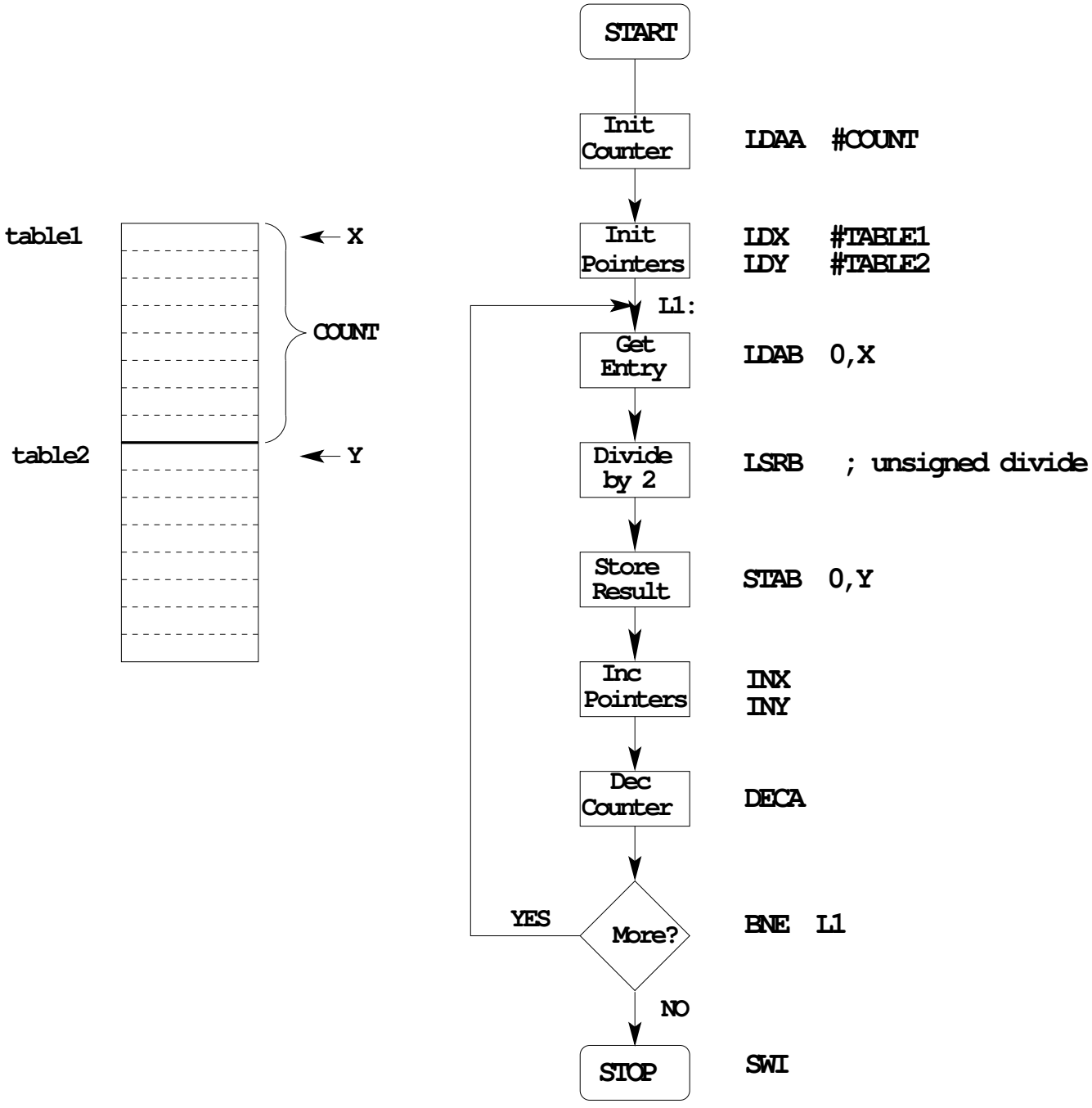
5. Use a simple flow chart to plan structure of program.



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     $2000
data:    equ     $1000

count:   equ     5

         org     prog     ;set program counter to 0x1000
         ldaa    #count   ;Use A 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
         bne     l1       ;counter != 0 => more entries to divide
         swi              ;Done


         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     $1000
data:    equ     $2000

count:   equ     5

         org     prog      ;set program counter to 0x1000
         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    1,x+      ;Get entry from table1; then inc pointer
         lsrb              ;Divide by two (unsigned)
         stab    1,y+      ;Save in table2; then inc pointer
         dbne    a,l1      ;Decrement counter; if not 0, more to do
         swi               ;Done


         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