Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C

Chapter 7
Structured Programming

Z. Gu

Fall 2025

Acknowledgement: Lecture slides based on Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C, University of Maine https://web.eece.maine.edu/~zhu/book/

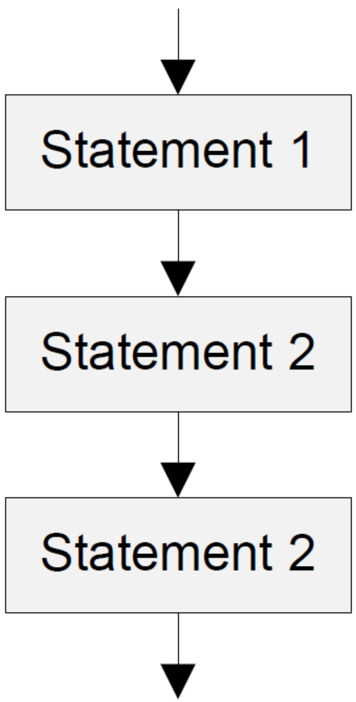
Divide and Conquer

"Nothing is particularly hard if you divide it into small jobs."

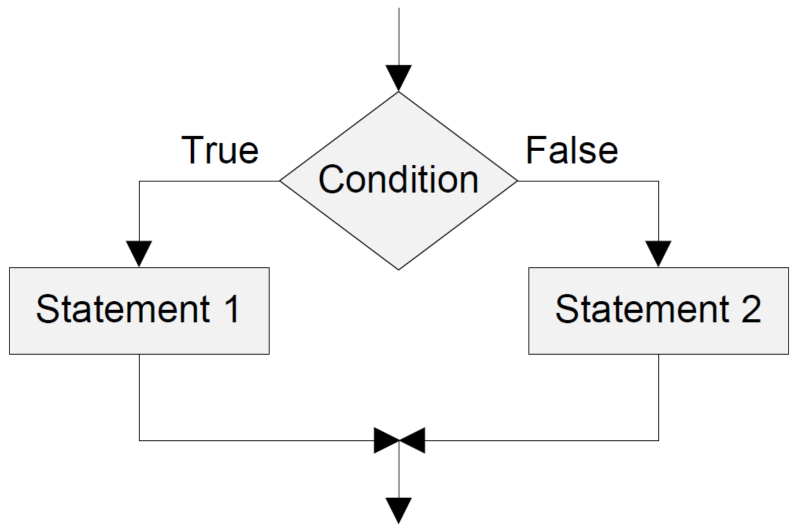
Henry Ford

Founder of Ford Motor

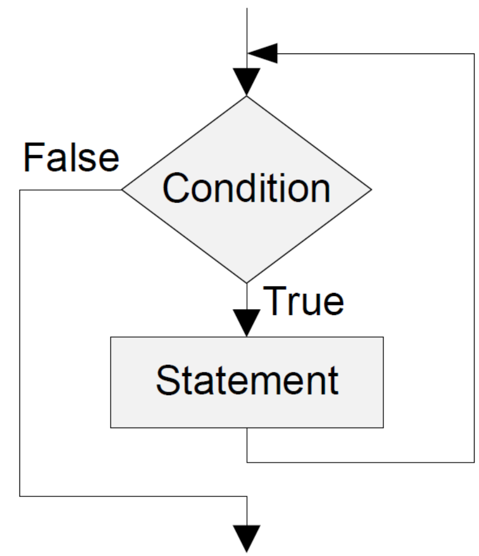
Basic Control Structures



Sequence Structure



Selection Structure



Loop Structure

History

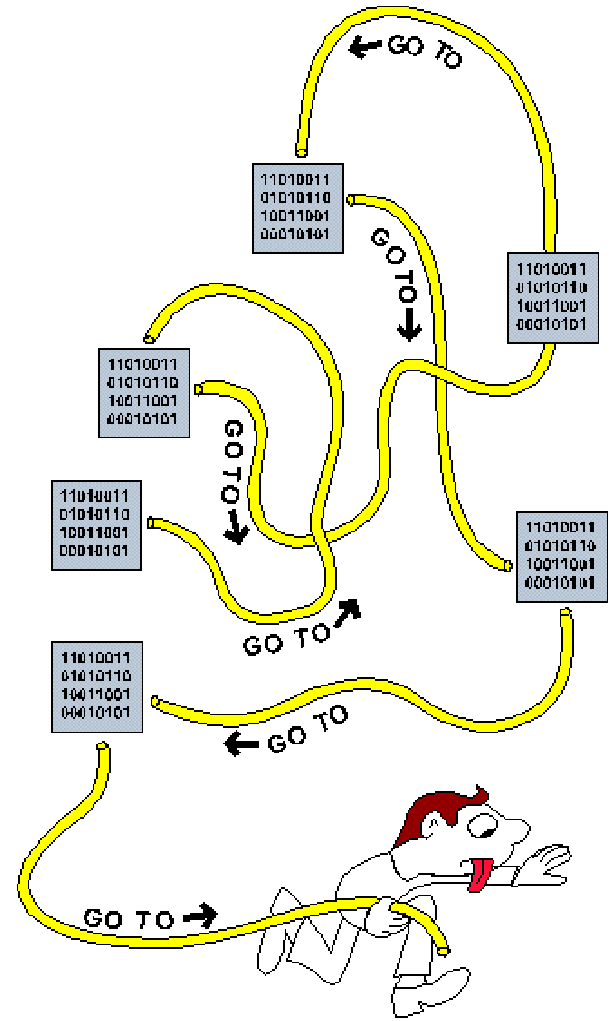
- Spaghetti Code
  - Before the 1980s, program flow bounces around anywhere the programmer wanted.
  - Culprit: overusing “GOTO” statements

Spaghetti code in BASIC

```
1 i=0
2 i=i+1
3 PRINT i; "squared=";i*i
4 IF i>=100 THEN GOTO 6
5 GOTO 2
6 PRINT "Program Completed."
7 END
```

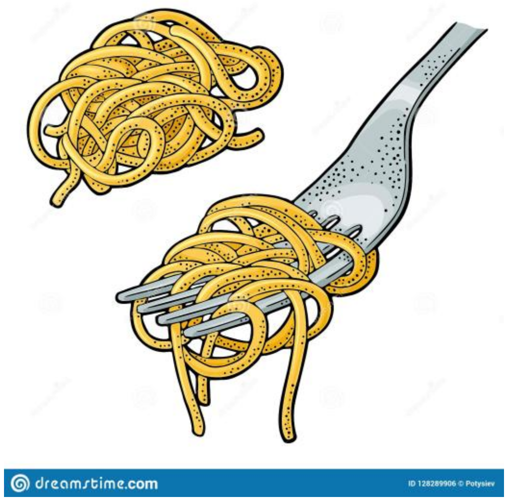
Structured programming in BASIC

```
1 FOR i=1 TO 100
2   PRINT i;"squared=";i*i
3 NEXT i    'termination of loop body
4 PRINT "Program Completed."
5 END
```



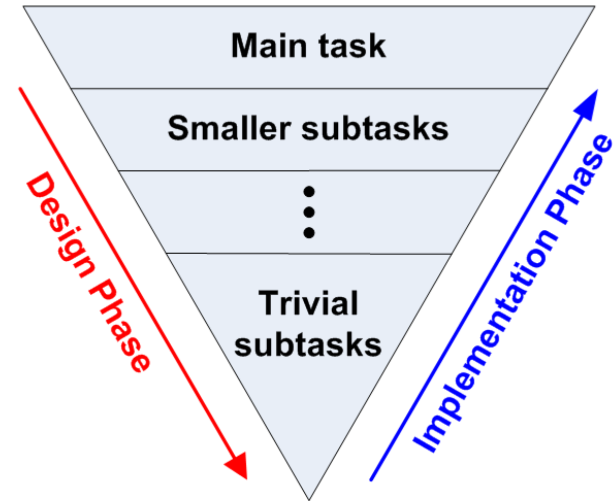
Importance of Structured Programming

- Assembly is not a structured programming language
  - Does not directly support selection and loop
  - Branch in assembly = "goto" in C
  - Break the single-entry single-exit rule
- Easy to generate spaghetti code in assembly
  - Twisted and tangled
  - Difficult to debug & maintain
- One strategy to alleviate the challenge
  - Use **flowcharts** to facilitate assembly programming
  - That is why textbook has many flowcharts
  - How to build flowcharts?



Software Design Strategy: Top-Down Design

- Three common design strategies
  - Top-down, also known as stepwise refinement
  - Bottom up
  - Object oriented
- Top-down: Repeatedly break down tasks into smaller and smaller pieces until they are easy to solve
- Example: Planning a picnic
  - Task 1: Where
  - Task 2: When
  - Task 3: Who
  - Task 4: Food



Top-Down Design Example

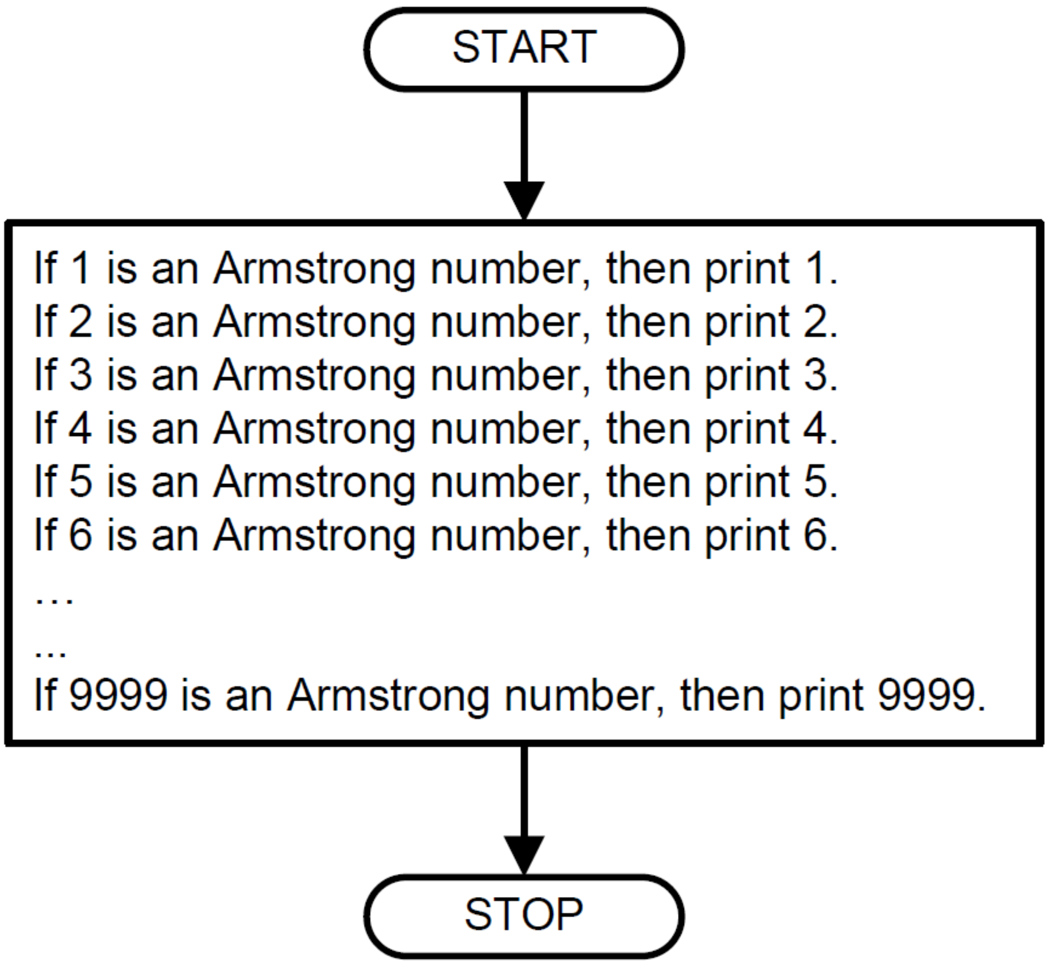
- Find all Armstrong numbers less than `10,000`
  - Given a positive integer that has *n* digits, it is an Armstrong number if the sum of the n^{th} powers of its digits equals the number itself.

$$153 = 1^3 + 5^3 + 3^3$$

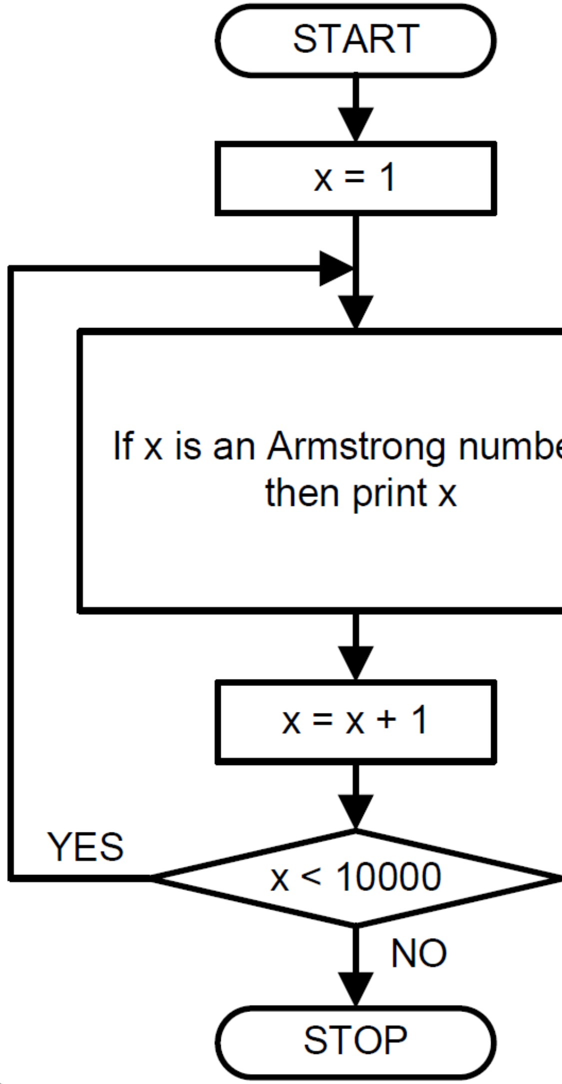
$$371 = 3^3 + 7^3 + 1^3$$

$$1634 = 1^4 + 6^4 + 3^4 + 4^4$$

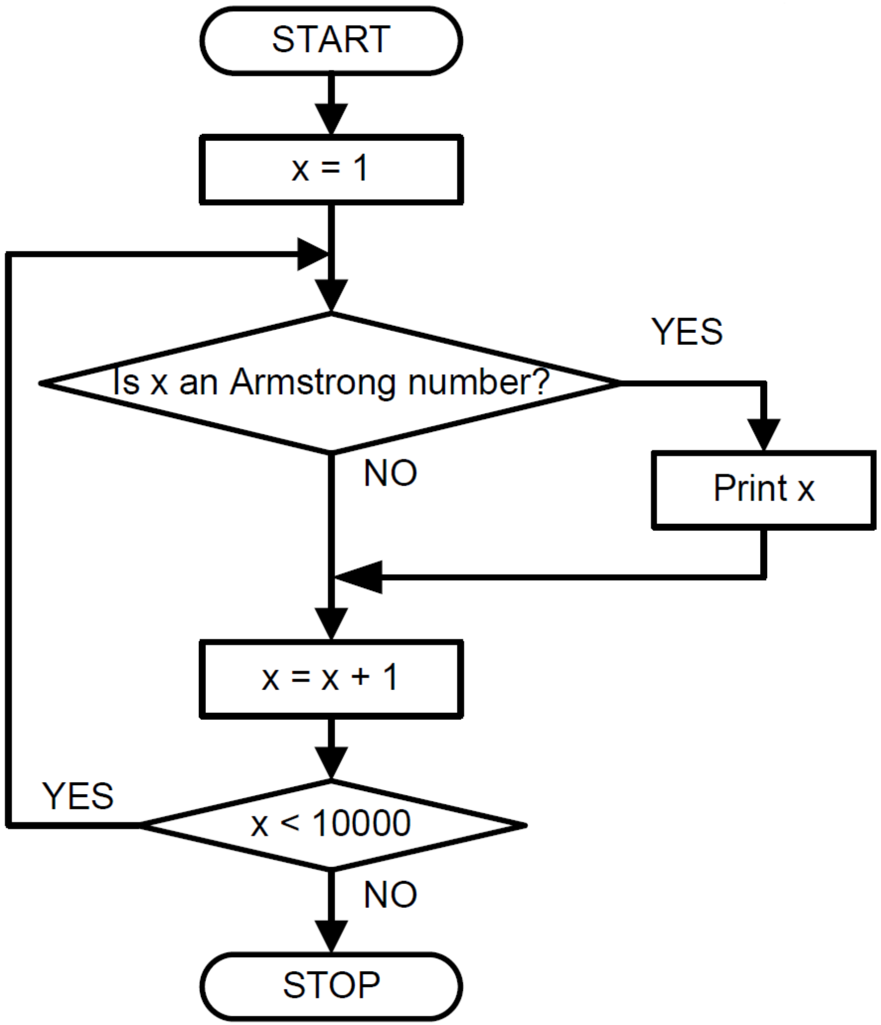
Top-Down Design Example



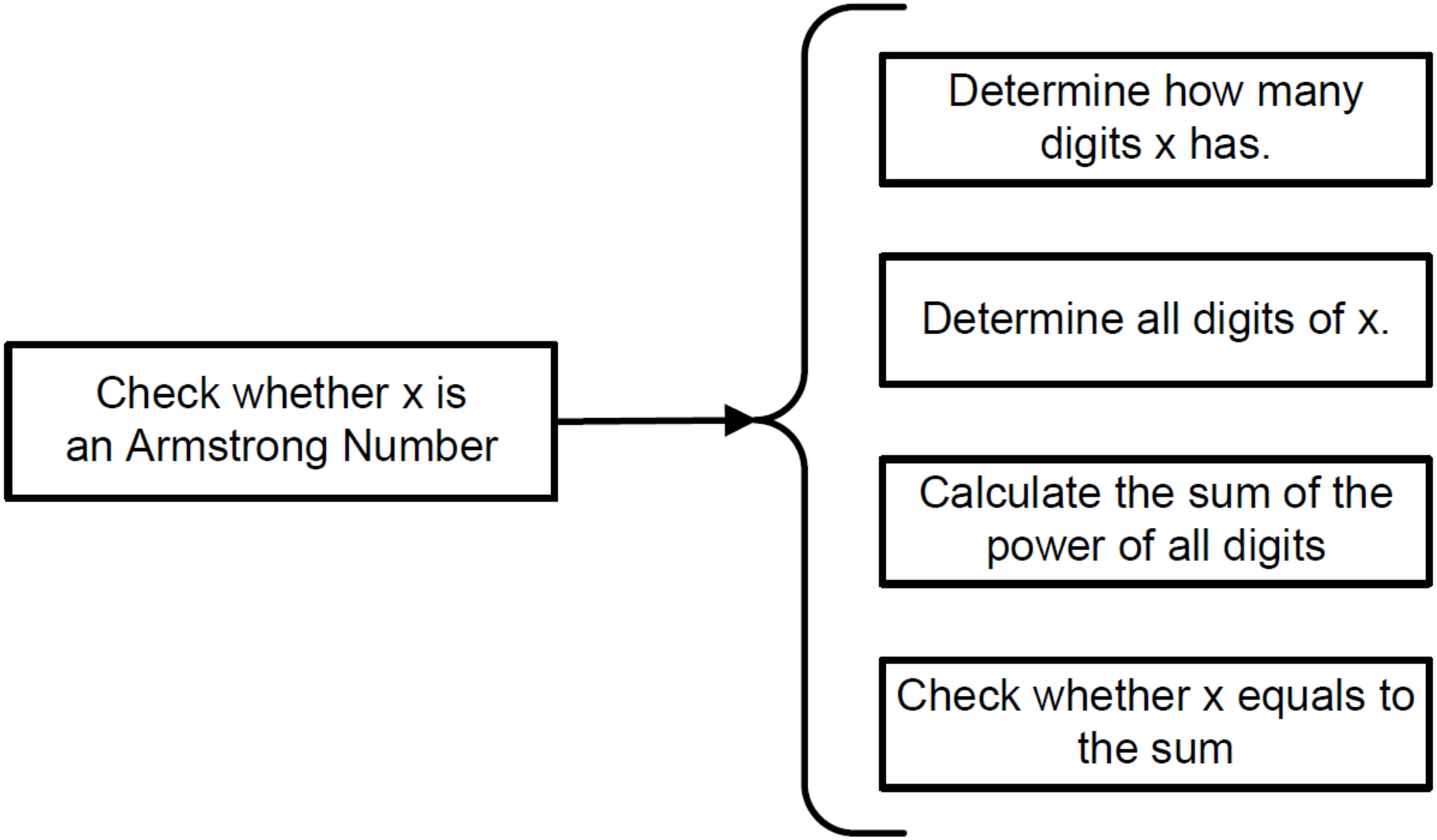
Top-Down Design Example



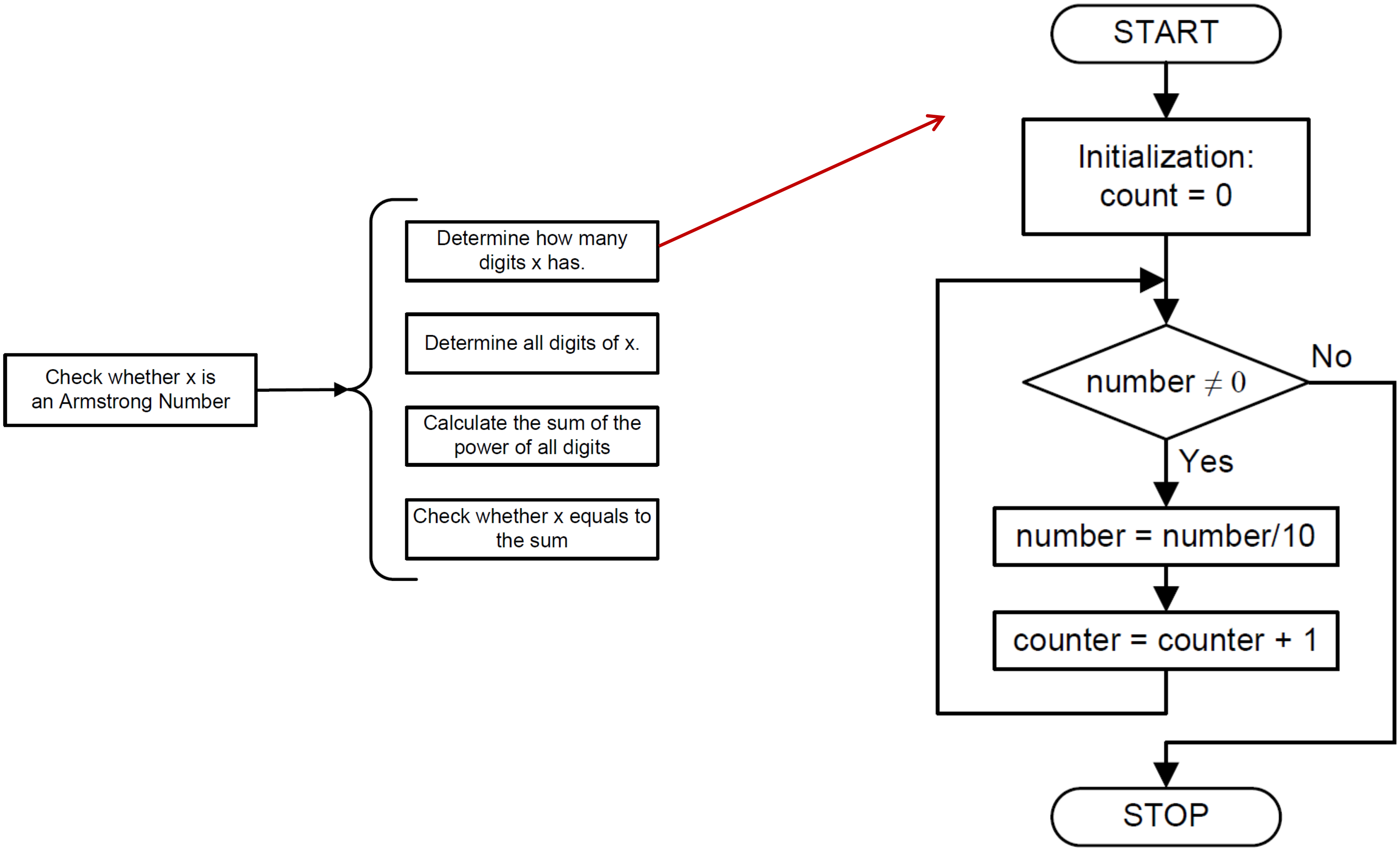
Top-Down Design Example



Top-Down Design Example



Top-Down Design Example:
Counting digits



Reuse Registers

```
int A = 0;  // 0x00000000
int B = -1; // 0xFFFFFFFF
int C = -2; // 0xFFFFFFFE
int D = 2;  // 0x00000002

void main(void){
  A = B + C – D;
  return;
}
```

Data memory

Address	***Data***	
0x2000,000F	0x00	D = 0x0000,0002 = 2
0x2000,000E	0x00	
0x2000,000D	0x00	
0x2000,000C	0x02	
0x2000,000B	0xFF	C = 0xFFFF,FFFE = -2
0x2000,000A	0xFF	
0x2000,0009	0xFF	
0x2000,0008	0xFE	
0x2000,0007	0xFF	B = 0xFFFF,FFFF = -1
0x2000,0006	0xFF	
0x2000,0005	0xFF	
0x2000,0004	0xFF	
0x2000,0003	0x00	A = 0x0000,0000 = 0
0x2000,0002	0x00	
0x2000,0001	0x00	
0x2000,0000	0x00	

Reuse Registers

```
int A = 0;  // 0x00000000
int B = -1; // 0xFFFFFFFF
int C = -2; // 0xFFFFFFFE
int D = 2;  // 0x00000002

void main(void){
  A = B + C – D;
  return;
}
```

Eight registers are used:
R0,r1,r2,r3,r4,r5,r6,r7

```
AREA myCode, CODE
   EXPORT __main
   ENTRY
__main PROC

   LDR r2, =B      ; r2 = 0x2000,0004
   LDR r3, [r2]    ; r3 = B = -1
   LDR r4, =C      ; r4 = 0x2000,0008
   LDR r5, [r4]    ; r5 = C = -2
   LDR r6, =D      ; r6 = 0x2000,000B
   LDR r7, [r6]    ; r7 = D = 2
   ADD r1, r3, r5  ; r1 = B + C
   SUB r1, r1, r7  ; r1 = B + C - D
   LDR r0, =A      ; r0 = 0x2000,0000
   STR r1, [r0]    ; Save A
   ENDP

   AREA myData, DATA
A  DCD  0
B  DCD  -1
C  DCD  -2
D  DCD  2

   END
```

Reuse Registers

```
        AREA myCode, CODE
        EXPORT __main
        ENTRY
__main PROC

        LDR r2, =B
        LDR r3, [r2]
        LDR r4, =C
        LDR r5, [r4]
        LDR r6, =D
        LDR r7, [r6]
        ADD r1, r3, r5
        SUB r1, r1, r7
        LDR r0, =A
        STR r1, [r0]
        ENDP

        AREA myData, DATA
A       DCD   0
B       DCD   -1
C       DCD   -2
D       DCD   2

        END
```

Lifetime of r3

8 registers used

```
        AREA myCode, CODE
        EXPORT __main
        ENTRY
__main PROC

        LDR r2, =B
        LDR r3, [r2]
        LDR r2, =C
        LDR r5, [r2]
        LDR r2, =D
        LDR r7, [r2]
        ADD r3, r3, r5
        SUB r3, r3, r7
        LDR r2, =A
        STR r3, [r2]
        ENDP

        AREA myData, DATA
A       DCD   0
B       DCD   -1
C       DCD   -2
D       DCD   2

        END
```

Lifetime of r2

Lifetime of r2

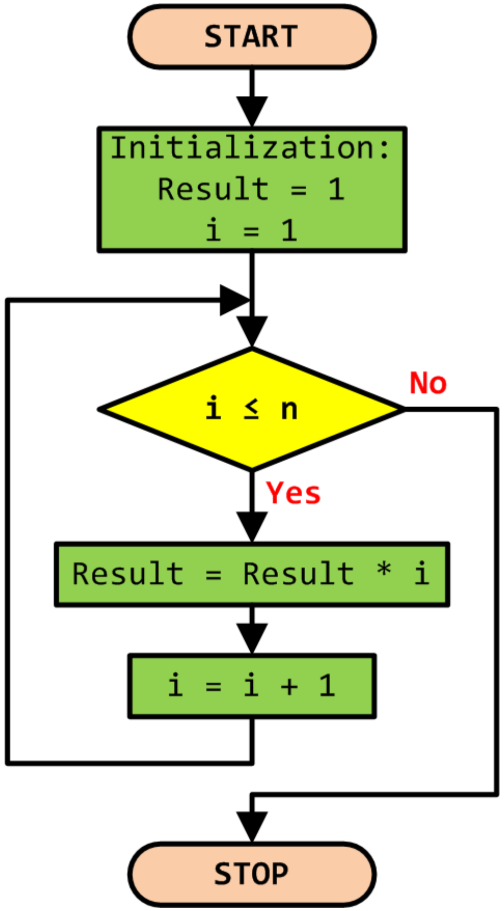
4 registers used

```
        AREA myCode, CODE
        EXPORT __main
        ENTRY
__main PROC

        LDR r2, =B
        LDR r3, [r2]
        LDR r2, =C
        LDR r5, [r2]
        LDR r2, =D
        LDR r2, [r2]
        ADD r3, r3, r5
        SUB r3, r3, r2
        LDR r2, =A
        STR r3, [r2]
        ENDP

        AREA myData, DATA
A       DCD   0
B       DCD   -1
C       DCD   -2
D       DCD   2

        END
```

Reuse r2

3 registers used

Example 1: Factorial Numbers



$$n! = \prod_{i=1}^{n} i = n \times (n-1) \times (n-2) \cdots \times 2 \times 1$$

Example 1: Factorial Numbers

C Program	Assembly Program
`` int main(void) { int result, n, i; ```	``` AREA factorial, CODE, READONLY EXPORT __main ENTRY __main PROC ```

```c
int main(void) {
  int result, n, i;
  result = 1;
  n = 5;

  for (i = 1; i <= n; i++)
    result = result * i;

  while(1);
}
```

```asm
       AREA factorial, CODE, READONLY
       EXPORT __main
       ENTRY
__main PROC
       MOV  r0, #1       ; r0 = result
       MOV  r1, #5       ; r1 = n

       MOV  r2, #1       ; r2 = i = 1
loop   CMP  r2, r1       ; compare i and n
       BGT  stop         ; if i > n, stop
       MULS r0, r2, r0   ; result *= i
       ADD  r2, r2, #1   ; i++
       B    loop

stop   B    stop
       ENDP
       END
```

Example 2: Counting Ones in a Word

```
Assembly Program
        AREA Count_Ones, CODE
        EXPORT __main
        ALIGN
        ENTRY
__main  PROC
        ; r0 = Input = x
        ; r1 = Number of ones = counter
        LDR  r0, =0xAAAAAAAA

        ; r1 = r0 >> 31
        MOV  r1, r0, LSR #31

        ; r0 = r0 << 2 and change Carry
loop    MOVS r0, r0, LSL #2

        ; r1 = r1 + r0 >> 31 + Carry
        ADC    r1, r1, r0, LSR #31
        BNE  loop

 stop     B        stop
        ENDP
        END
```

After LDR: r0=10101010101010101010101010101010 (Load input data into r0)

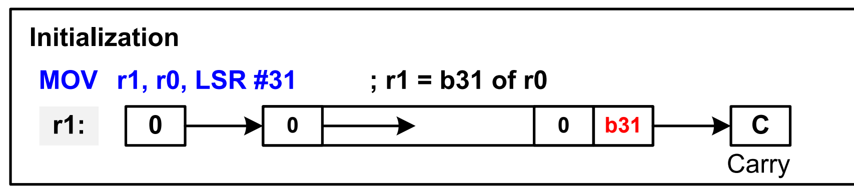
After MOV: r1=r0 >> 31 = 1 (Initialize r1 with the most significant bit of r0. r0 logical shift right by 31 bits, take the leftmost bit b31)

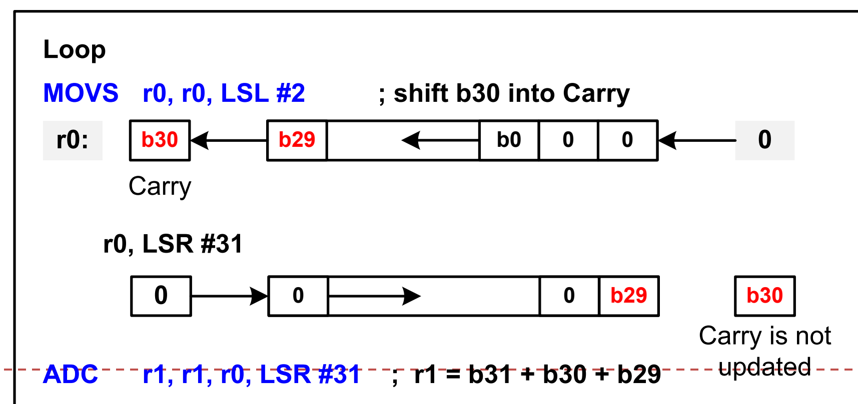
After MOVS: r0 = r0 << 2 =10101010101010101010101010101000 (Logical shift left r0 by 2 bits and update C = 0, as the last shifted out bit b30)

After ADC: r1 = r1 + r0 >> 31 + Carry = b31 + b29 + b30 = 1 + 1 + 0 = 2

2nd iteration: r1 = r1 + b28 + b27 = 2 + 1 + 0 = 3

If after MOVS, the result in r0 is zero, (no more 1's), Z flag is set to 1 and the loop exits





At the end of the first loop: **r1 = b31 + b30 + b29**

Example 2: Counting Ones in a Word: Explanations

Iteration	Shifted r0 value (MSB bit)	Carry bit (last shifted out)	r1 (accumulated count)	Notes
0 (init)	—	—	b31 = 1	r1 initialized with b31
1	b29 = 1	b30 = 0	1 (b31) + 1 (b29) + 0 (b30) = 2	r0 shifted left by 2 bits
2	b27 = 1	b28 = 0	2 + 1 (b27) + 0 (b28) = 3	
3	b25 = 1	b26 = 0	3 + 1 (b25) + 0 (b26) = 4	
4	b23 = 1	b24 = 0	4 + 1 (b23) + 0 (b24) = 5	

- in r0 0xAAAAAAAA, bits at odd positions (31, 29, 27, ..., 1) are all 1, bits at even positions (30, 28, 26, ..., 0) are all 0.
- Carry bit is always the even bit index at each iteration.
- At each iteration, r1 accumulates 1 (highest bit, odd index) + 0 (carry bit, even index).
- The loop ends when r0 becomes zero after the last shift, triggering the Zero flag and exiting the branch.
- The count accumulates to 16, consistent with the fact that 0xAAAAAAAA has exactly 16 ones in 32 bits.
- This program counts two bits per loop iteration, leveraging the Carry bit, and will take 16 iterations for a 32-bit word.

Stop B stop

- “stop B stop” means an infinite loop that repeatedly branches to the label "stop".
  - B is the branch instruction in ARM, which causes the program to jump to the specified label or address.
  - Here, the label and destination are both "stop". This creates a loop where execution never moves past this point.
  - It is commonly used to halt the program or wait indefinitely, often when the program completes or to prevent it from running into uninitialized memory.

Example 2: Counting Ones in a Word: Simpler Programs

Algo 1

```
    LDR r0, =0xAAAAAAAA      ; Load input data into r0
    MOV r2, #0               ; Initialize count (r2) to 0

loop:
    MOV r1, r0, LSR #31      ; Extract leftmost bit of r0 into r1 (0 or 1)
    ADD r2, r2, r1           ; Add extracted bit to count in r2
    MOVS r0, r0, LSL #1      ; Shift r0 left by 1 bit, update flags
    BNE loop                 ; If r0 != 0, repeat loop
```

Algo 2

```
    LDR r0, =0xAAAAAAAA      ; Load input data into r0
    MOV r2, #0               ; Initialize bit count accumulator (r2) to 0

loop:
    MOVS r0, r0, LSL #1      ; Shift left by 1 bit, carry gets old MSB
    ADC r2, r2, #0           ; Add carry (0 + carry) to r2
    BNE loop           ; Loop while r0 != 0 (Zero flag clear)
```

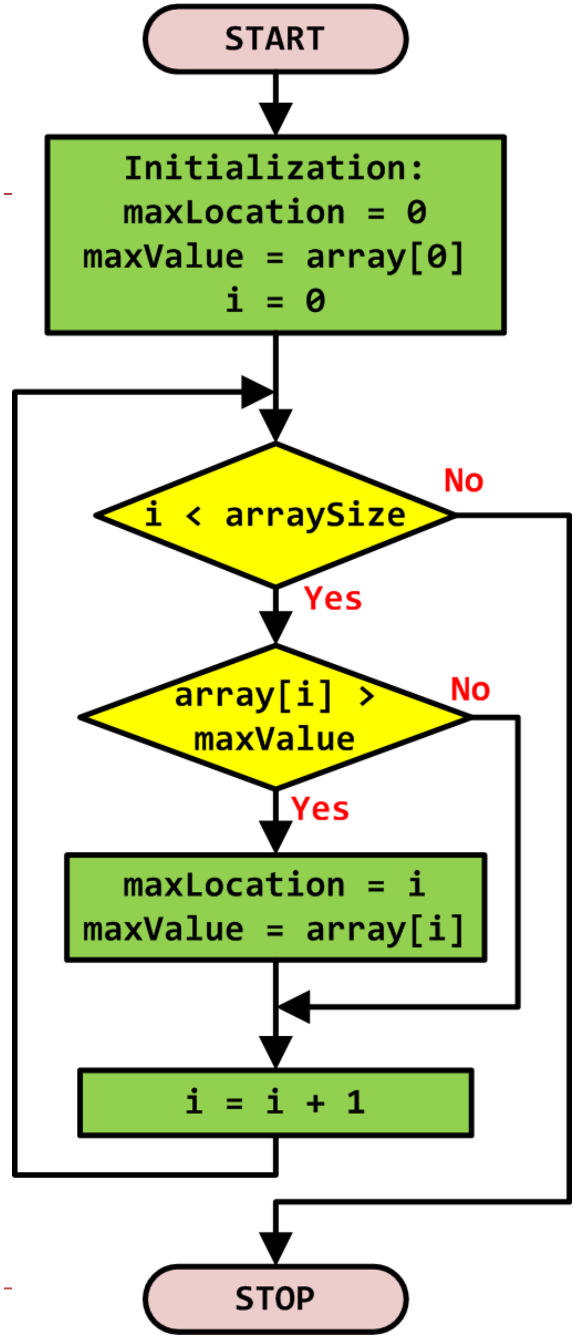
- Algo 1: use MOV r1, r0, LSR #31 to extract the highest bit from r0 and accumulates the per-bit count. (Carry flag is set but ignored.)
- Algo 2: use MOVS r0, r0, LSL #1 to shift left by 1 bit and update the carry flag with the bit shifted out (the leftmost bit of the original value). Use ADC (Add with Carry) to add the carry bit to accumulator r2 without needing to move the leftmost bit explicitly.
- Both programs count one bit per loop iteration, discarding the Carry bit, and will take 32 iterations for a 32-bit word.

Quiz

- Q: In Algo 1 and Algo 2, can we change MOVS r0, r0, LSL #1 to MOV r0, r0, LSL #1?
- ANS:
- Q: In Algo 1, can we change MOV r1, r0, LSR #31 to MOVS r1, r0, LSR #31
- ANS:
- Q: In Algo 1, can we change ADD r2, r2, r1 to ADC r2, r2, r1
- ANS:
- Q: In Algo 2, can we change ADC r2, r2, #0 to ADD r2, r2, #0?
- ANS:

Example 3: Finding Max of an Array

```
// Initialize max and location
maxLocation = 0;
maxValue = array[0];

// Loop through the array
for (i = 0; i < arraySize; i++){
    if (array[i] > maxValue) {
        maxValue  = array[i];
        maxLocation = i;
    }
}
```



Example 3: Finding Max of an Array

C Program	Assembly Program
`int array[10] = {-1, 5, 3, 8, 10, 23, 6, 5, 2, -10};` `int size = 10;`	`        AREA myData, DATA` `        ALIGN` `array   DCD -1,5,3,8,10,23,6,5,2,-10` `size    DCD 10`
`int main(void) {` `  int i, maxLocation, maxValue;`	`        AREA findMax, CODE` `        EXPORT __main` `        ALIGN` `        ENTRY` `__main  PROC` `        ; Identify the array size` `        LDR   r3, =size` `        LDR   r3, [r3]     ; array size` `        SUB   r3, r3, #1`
`  // Initialize max and location` `  maxLocation = 0;` `  maxValue = array[0];`	`        ; Initialize max value and location` `        LDR   r4, =array` `        LDR   r0, [r4]     ; r0 = default max` `        MOV   r1, #0       ; r1 = max location`
`  // Loop through the array` `  for (i = 0; i < size; i++){` `    if (array[i] > maxValue) {` `        maxValue  = array[i];` `        maxLocation = i;` `    }` `  }`	`        ; Loop over the array` `        MOV   r2, #0       ; loop index i` `loop    CMP   r2, r3       ; compare i & size` `        BGE   stop         ; stop if i ≥ size` `        LDR   r5, [r4,r2,LSL #2]   ; array[i]` `        CMP   r5, r0       ; compare with max` `        MOVGT r0, r5       ; update max value` `        MOVGT r1, r2       ; update location` `        ADD   r2, r2, #1   ; update index i` `        B     loop`
`  while(1); //dead loop` `}`	`stop    B     stop         ; dead loop` `        ENDP` `        END`

Quiz

- Instruction LDR r5, [r4,r2,LSL #2] has the form:
- LDR <destination register>, [<base register>, <index register>, LSL #<shift amount>]
- Address=value in r4+(value in r2$\times 2^2$)
  - The memory address is calculated by taking the value in the <base register> (here r4) plus the value in the <index register> (here r2) shifted left (logical shift left, LSL) by a certain number of bits (#2 means shifted by 2 bits, or multiplied by 4).
- Q: why do we perform r2$\times 2^2$ here?
- ANS: