# Microcomputers I – CE 320

Mohammad Ghamari, Ph.D.
Electrical and Computer Engineering
Kettering University

# Announcements

- Do not forget that you have a quiz on Friday!

- Homework Exercise 3 is added on blackboard.

# Lecture 9: Structured Assembly Program Design

# Today’s Goals

By the end of this class you should be able to:

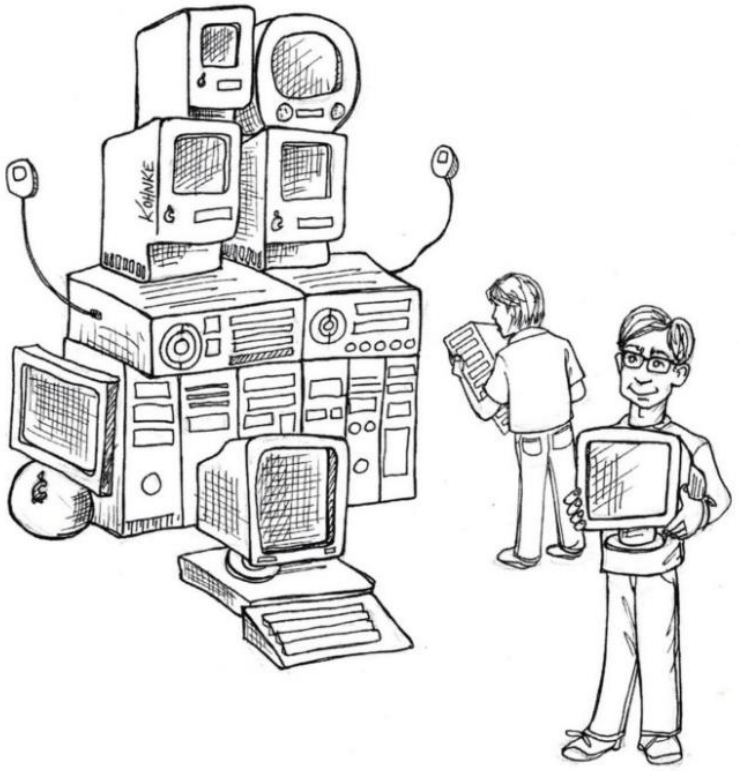
- Explain the advantages of structured programming.
- Identify the main flowcharting constructs.
- Generate structured flowcharts from program requirements.
- Translate a flowchart into assembly program.

# What is structured programming?

- A style of programming that has proven to be very useful in designing programs.

- It's based on expressing a program's function in terms of a few *basic* ***programming structures***, or ***constructs***.

- The key feature to a construct is that it has **only *one entry point*** and **only *one exit point***. This property is critical for maintaining the structured nature.



# Structured Programming

- Some of the main advantages of structured programming are:

1. Easy to understand, especially to those who did not write the program initially.
2. Easy to modify/update.
3. Often more straightforward to verify correct operation.
4. Avoids spaghetti programs, which have a needlessly high amount of branching.

# Flowcharts

- In this course, we will primarily use ***flowcharting*** to represent structured programs.

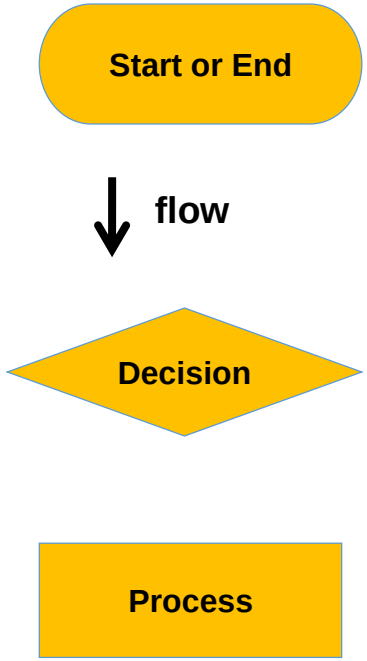
- There are other methods of representing programs, such as state transition diagrams and UML (Unified Modeling Language).

- However flowcharting works well for software written at the assembly code level.

# Flowcharts

## Meaning of symbols

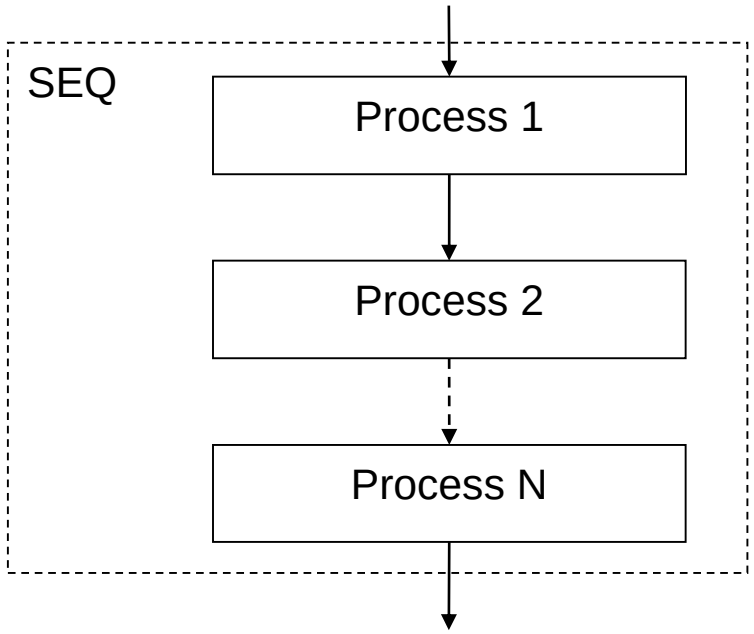
- There are many more symbols other than these.
- Shapes may be different.
- In this course, those four symbols are pretty much all we need.



# Construct: Sequence

- One or more process blocks strung together serially.

**Flowchart Construct:**




**Assembly Code Template:**

Code for Process 1

Code for Process 2

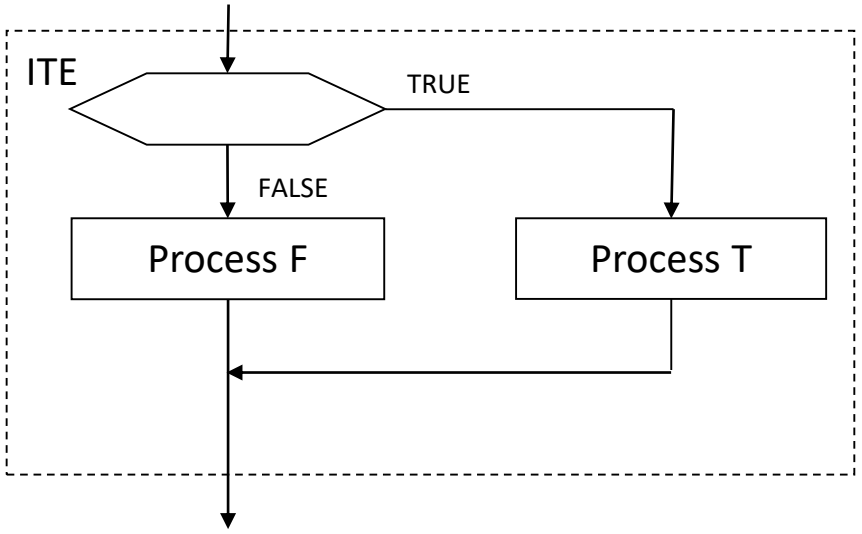
…

Code for Process N

# Construct: If-Then-Else

- One process is executed if a condition evaluates to TRUE, and another process is executed if the condition is FALSE.
- Note that one of the processes could be NULL, and code is usually more efficient if the statement is phrased so that the TRUE process is NULL.

## Flowchart Construct:




## Assembly Code Template:

Set CCR bits for decision

Bxx Process T

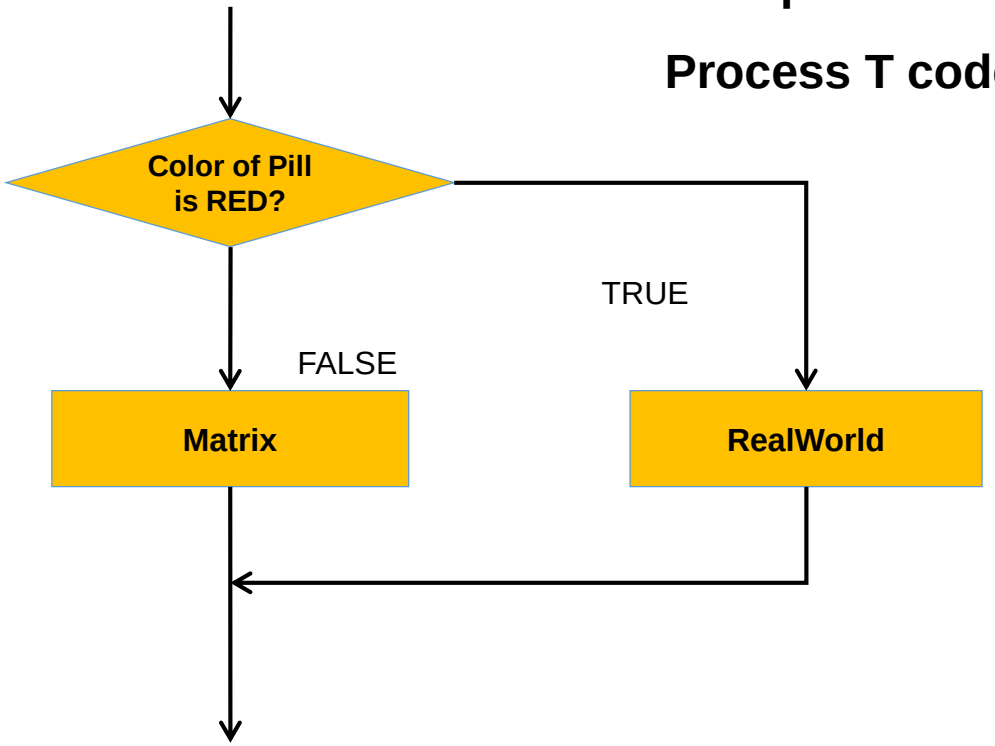
Process F code

BRA past Process T code (optional)

Process T code (optional)

# If-Then-Else

Example:




Assembly Code Template:

**Set CCR bits for decision**

**Bxx Process T**

**Process F code**

**BRA past Process T code (optional)**

**Process T code (optional)**

**If(PillColor == RED)**
**  RealWorld**
**Else**
**  Matrix**

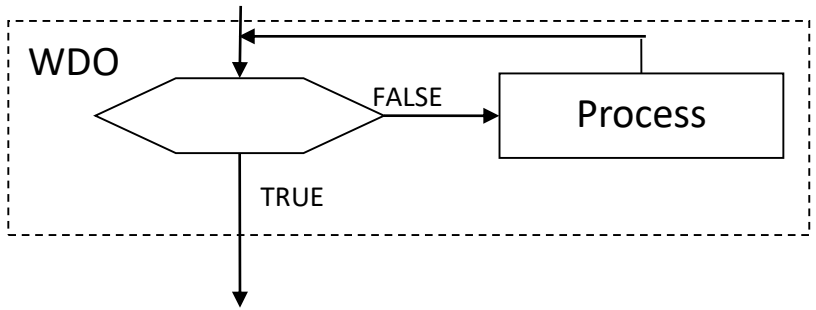
Example:

```
        ldaa    PillColor
        cmpa    #RED
        bne     Imatrix
            RealWorld
            bra     Iskip
Imatrix:    Matrix
Iskip:  ...
```

# Construct: While Do

- Evaluate an exit statement. If the statement is FALSE, execute a process and reevaluate the exit statement.

Flowchart Construct:




Assembly Code Template:

Affect CCR bits for Decision

Bxx past BRA

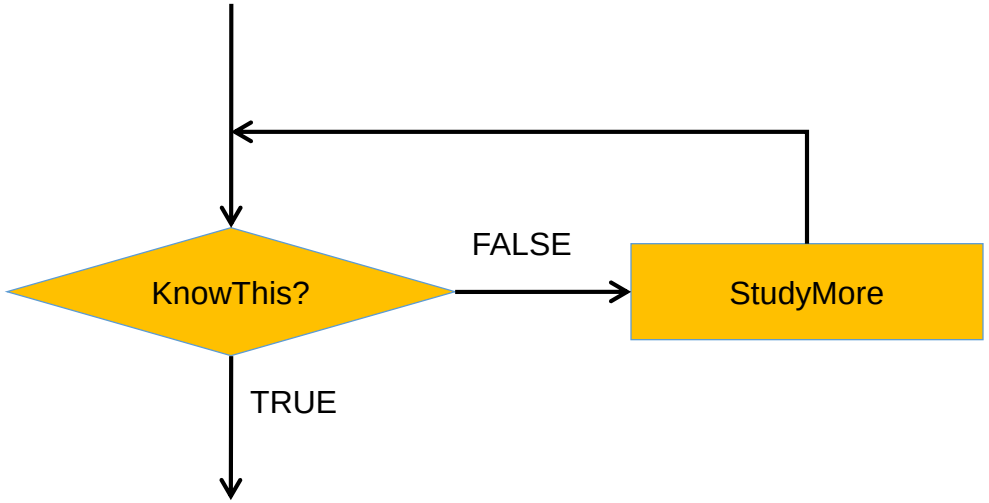
Process code

Affect CCR bits for Decision

BRA to Bxx

# While Do

Example:




**Assembly Code Template:**

Set CCR bits for decision
Bxx past BRA
Process  code
Set CCR bits for decision
BRA to Bxx

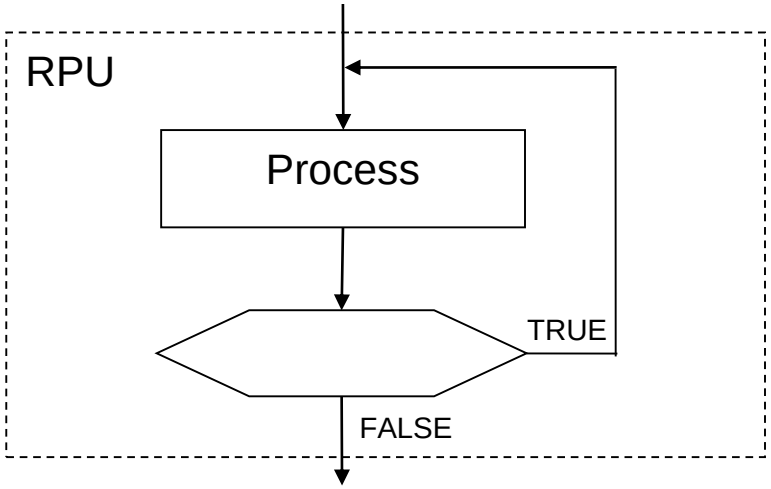
Example:

```
        ldaa    KnowThis
lloop:  cmpa    #YES
        beq     lnext
        StudyMore
        ldaa    KnowThis
        bra     lloop
lnext:  NextStep
```

# Construct: Repeat Until

- Execute a process block. Evaluate a repeat statement. If the statement is TRUE, branch backwards to the process block.

Flowchart Construct:




Assembly Code Template:

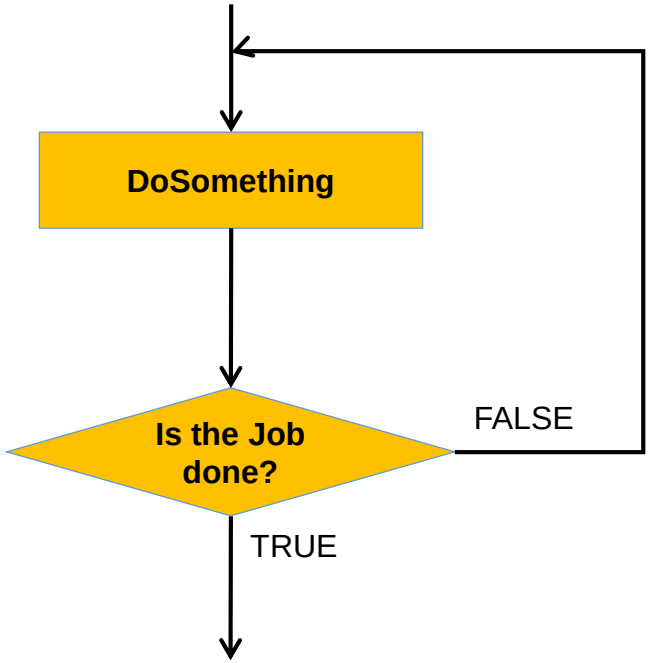
Process code

Affect CCR Bits

Bxx to Process code

# Repeat Until

Example:




**Assembly Code Template:**

Process code
Set CCR bits
Bxx to Process code

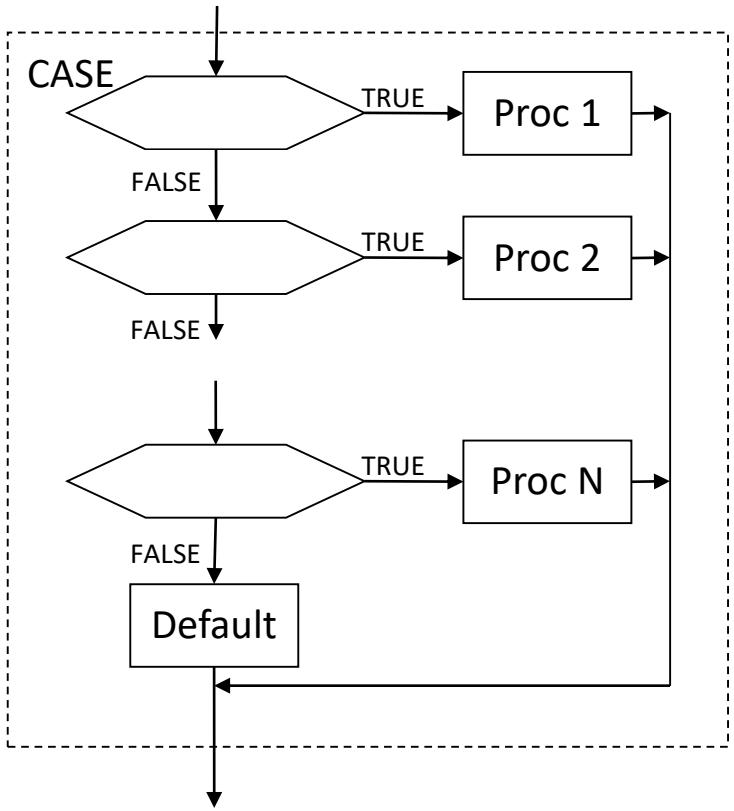
Example:

```
loop:   DoSomething
        ldaa    Result
        beq     ldone
        bra     loop
ldone:
```

# Construct: Case

- This is basically a group of nested If-Then-Else constructs.
- Only one from a set of processes will be executed.

Flowchart Construct:

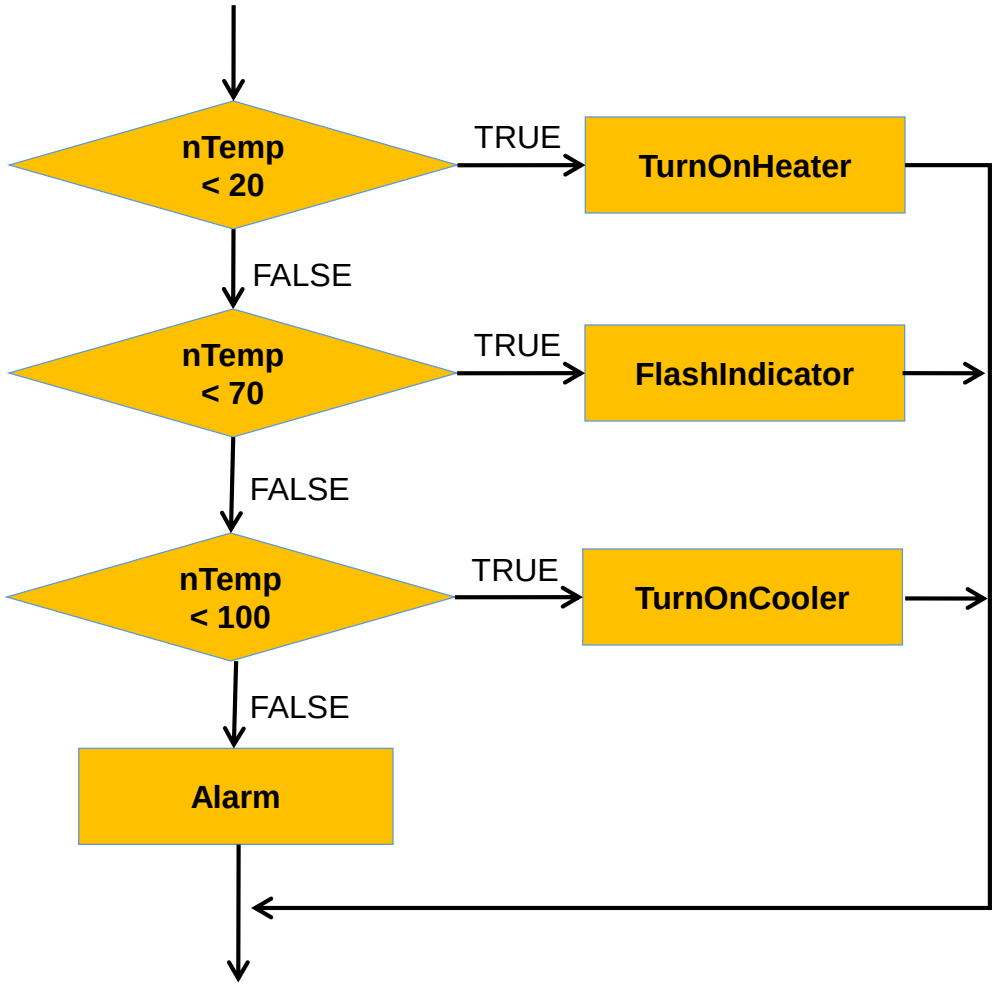



Assembly Code Template:

Affect CCR bits
Bxx Process 1
Affect CCR bits
Bxx Process 2
…
Affect CCR bits
Bxx Process N
Default Process Code
BRA past Process N code
Process 1 code
BRA past Process N code
Process 2 code
BRA past Process N code
…..
Process N code

# Case

## Example:




Set CCR bits
Bxx to Process 1
Set CCR bits
Bxx to Process 2
…
Set CCR bits
Bxx to Process N
Default Process code
BRA past Process N code
Process 1 code
BRA past process N code
Process 2 code
BRA past Process N code
…
Process N code

Example:

```
            ldaa    nTemp
            cmpa    #20
            bhs             l1
            TurnOnHeater
            bra             lres
l1:         cmpa    #70
            bhi             l2
            FlashIndicator
            bra             lres
l2:         cmpa    #100
            bhi             l3
            TurnOnCooler
            bra             lres
l3:         Alarm
lres:
```

# Flowchart Guidelines

- The following guidelines help flowcharts be more useful as an ***aid to a programmer*** as opposed to something that is drawn after a program already works as part of a lab report.

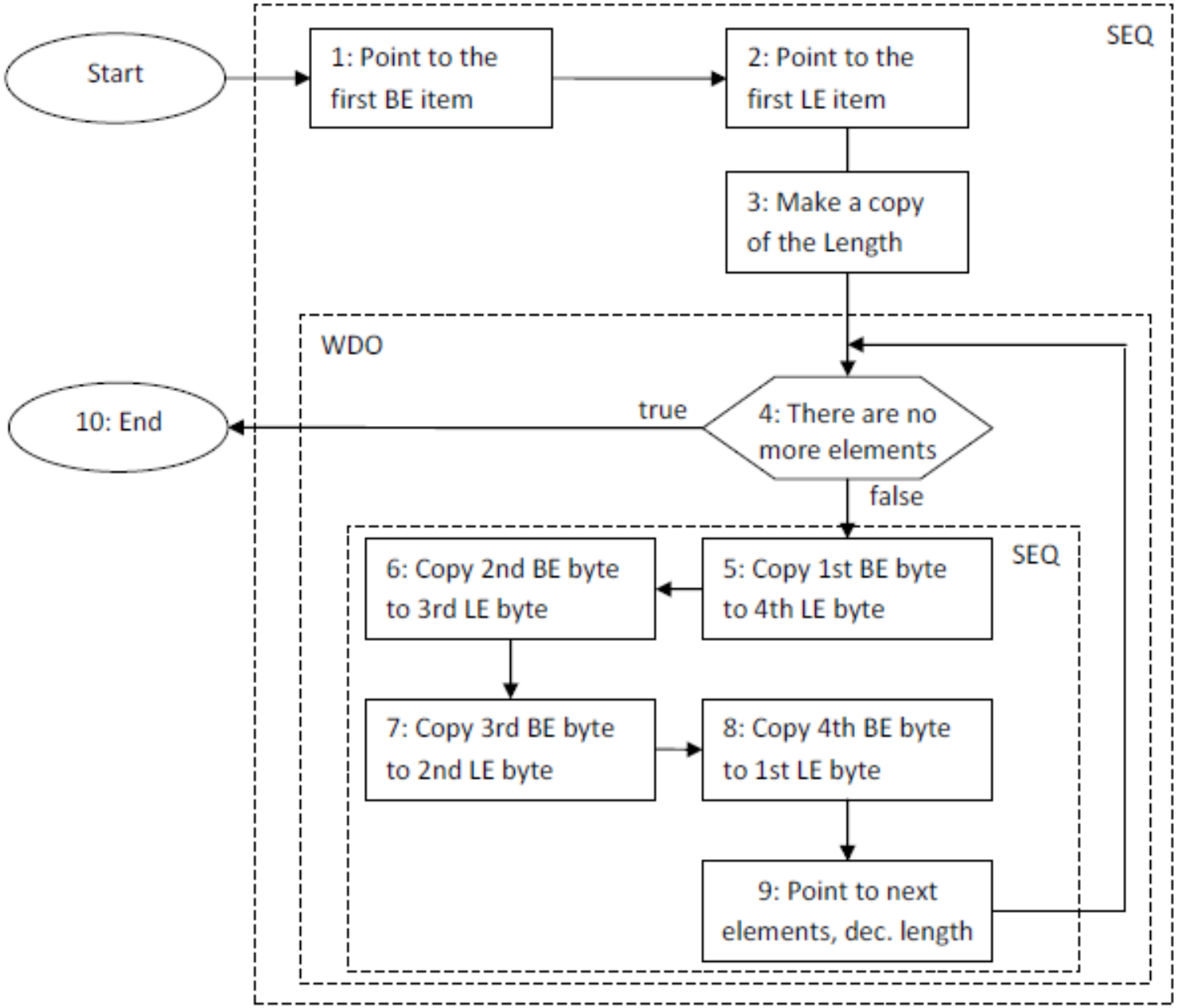
1. Do not refer to registers in the flowchart.

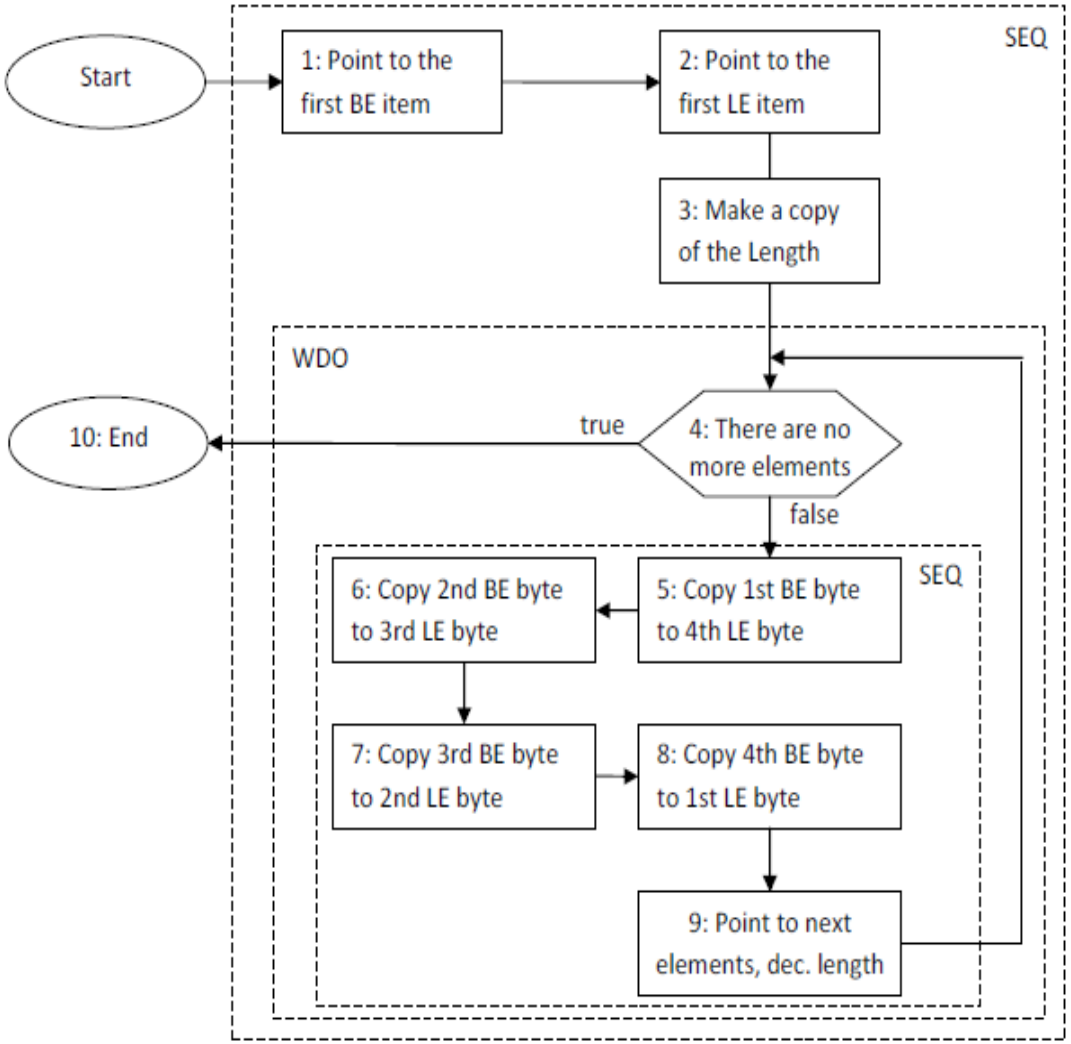
2. Arrows should never cross (they will not need to if the flowchart represents a structured program).

3. The purpose is to remove any questions about how to program and understand the algorithm, and this usually determines when the ***flowchart contains enough detail***.

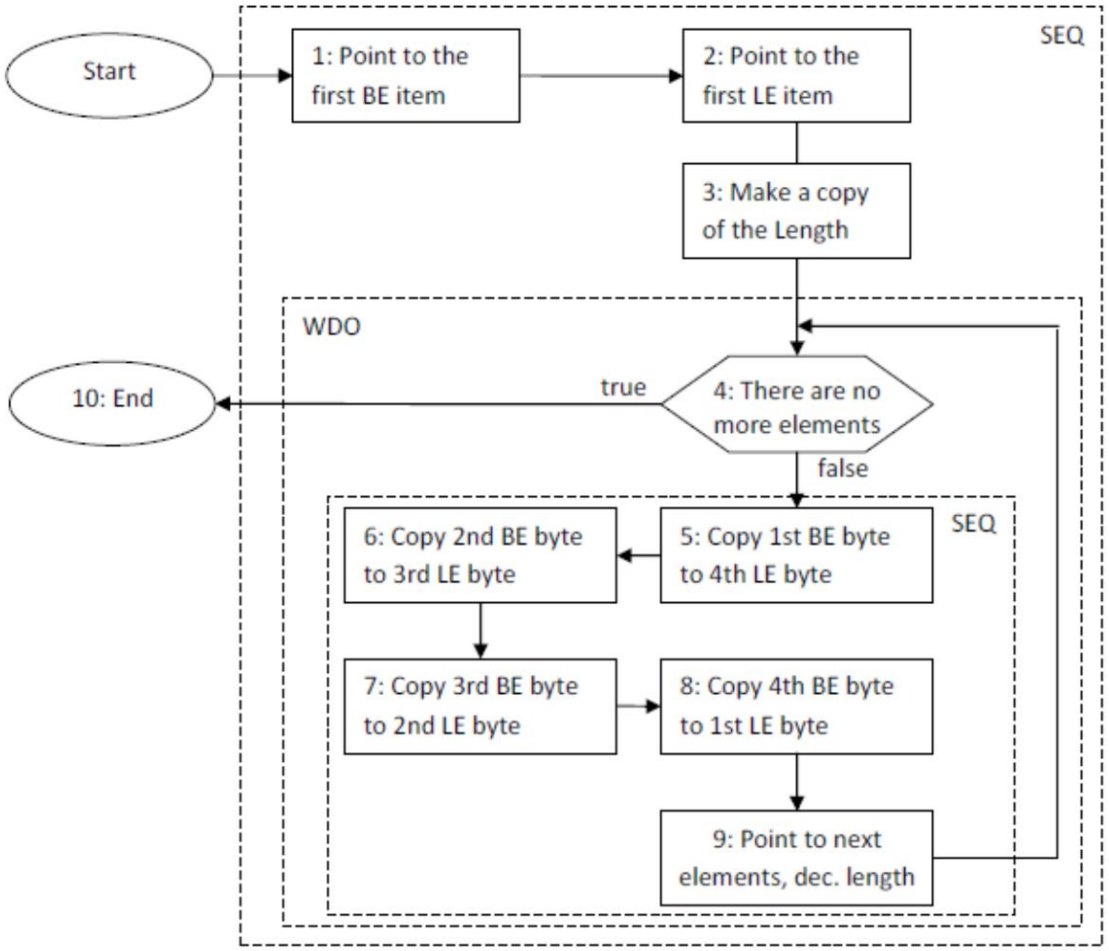
# Homework Example

- Design a flowchart for a program that converts an array of 4-byte Big-Endian values to an array of Little-Endian values. Assume:
  - Address $3000 holds the address of the array of Big-Endian values,
  - Address $3002 holds the address of the array for the Little-Endian values, and
  - Address $3004 holds the two-byte length of numbers to convert (not bytes of the table).
- Write an assembly program that implements this.


SEQ
Start
1: Point to the first BE item
2: Point to the first LE item
3: Make a copy of the Length
WDO
10: End
true
4: There are no more elements
false
SEQ
6: Copy 2nd BE byte to 3rd LE byte
5: Copy 1st BE byte to 4th LE byte
7: Copy 3rd BE byte to 2nd LE byte
8: Copy 4th BE byte to 1st LE byte
9: Point to next elements, dec. length




```
                org  $3000
BEnd            ds.w  1
LEnd            ds.w  1
Length          ds.w  1
TmpLen          ds.w  1
MyCode:   SECTION
                ldx  BEnd               ;(1)3
                ldy  LEnd               ;(2)3
                ldd  Length             ;(3)3
Loop            std  TmpLen             ;(3,4,9)3
                beq  Done               ;(4)1/3
                ldaa  0,x               ;(5)3
                staa  3,y               ;(5)2
                ldaa  1,x               ;(6)3
                staa  2,y               ;(6)2
                ldaa  2,x               ;(7)3
                staa  1,y               ;(7)2
                ldaa  3,x               ;(8)3
                staa  0,y               ;(8)2
                inx                     ;(9)1
                inx                     ;(9)1
                inx                     ;(9)1
                inx                     ;(9)1
                iny                     ;(9)1
                iny                     ;(9)1
                iny                     ;(9)1
                iny                     ;(9)1
                ldd   TmpLen            ;(9)3
                subd  #0001             ;(9)2
                bra  Loop               ;(9)3
Done            swi                     ;(10)9
```




```
;Here is a more efficient program
            org  $3000
BEnd        ds.w  1
LEnd        ds.w  1
Length      ds.w  1
MyCode:     SECTION
            ldx  BEnd               ;(1)3
            ldy  LEnd               ;(2)3
            ldd  Length             ;(3,4)3
Loop        beq  Done               ;(4)1/3
            movb  0,x,3,y           ;(5)5
            movb  1,x,2,y           ;(5)5
            movb  2,x,1,y           ;(5)5
            movb  3,x,0,y           ;(5)5
            leax  4,x               ;(9)2
            leay  4,y               ;(9)2
            subd  #$0001            ;(9)2
            bra  Loop               ;(9)3
Done        swi                     ;(10)9
```