An Introduction to Z80 Microprocessor Applications

DT202 Curriculum Manual

Lesson Module: 8.22/1
Issue: MP612/A

About this Module

For tomorrow's engineers and technicians, training in the use of microprocessor systems and the design of control tasks will be very important.

We see microprocessors used in almost every area of modern life. They control domestic appliances, automated Teller machines, VCRs, automobile engine management and braking systems and so on - the applications are endless. In addition to these less obvious uses, microprocessors dominate today's' working environment in the shape of the PC (personal computer).

To gain a good working knowledge of microprocessor technology you will need to follow this manual carefully. It will lead you in a step by step manner through the following areas:

- Using the SAM microcomputer.
- Introduction to Z80 programming.
- Writing Machine Code Programs.
- Program Debugging.
- Using the Merlin Text Editor.
- Introduction to Development Systems.
- Addressing Modes.
- Negative Binary Numbers.
- Programs with Loops.
- Further Programs with Loops.
- Logical and Test Instructions.
- Input and Output Programming.
- Programming the Applications Module.
- Stack and Subroutines.
- Interrupts.

As you work through each chapter you will be guided by a series of student objectives and your progress will be continually assessed by questions in the Exercises, Practical Assignments and Student Assessments.

The Practical Assignments presented throughout the manual are graded in terms of complexity, starting with simple machine code programs and ending with more complex programming techniques in assembler code.

Your instructor has a copy of the Solutions book for this manual. It contains all the solutions to the assessment questions together with suggested solutions to all the programming tasks. Copies of these programs are provided on a disk supplied with the Solutions book.

What do I need to work through this manual?

To work through this manual you will need the following items:

1. SAM Z80 microprocessor board.
2. Merlin Development System software pack (6502/Z80 version) and RS232 cable.
3. Microprocessor Applications board.
4. Personal computer (PC) running Windows 95 or later, and fitted with RS232 serial communications (COM) port.
5. Two 0.1" shorting leads (supplied).
6. SAM Z80 User Manual.
7. Z80 Programming Manual.
8. Note pad and pencil.

In addition, you will need a **power supply** and a **keypad/display unit**. The form that these items take will depend on whether you are using a *Digiac 2000* system or a *Digiac 3000* system:

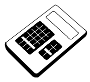
	Power supply required	**Keypad/display unit required**
Digiac 2000 system	DT60 Power Supply unit	DT25 Keypad/display module
Digiac 3000 system	D3000 Experiment Platform or D3000 Virtual Instrument Platform	D3000-8.0 Microprocessor Master Board with built-in keypad/display

For further information, please refer to the SAM Z80 User Manual.

Computerized Assessment of Student Performance

If your laboratory is equipped with the ***D3000*** Computer Based Training System, then the system may be used to automatically monitor your progress as you work through this manual.

If your instructor has asked you to use this facility, then you should key in your responses to the questions in this manual at your computer managed workstation.

To remind you to do this, a  symbol is printed alongside questions that require a keyed-in response.

The following D3000 Lesson Module is available for use with this manual:

D3000 Lesson Module 8.22

Additional Teachware

If you are encountering microprocessors for the first time, it is recommended that you begin by reading the manual "An Introduction to Microprocessor Technology", which is available from LJ Technical Systems.

Other manuals available in this range are:

An Introduction to 6502 Microprocessor Applications.
An Introduction to 6502 Microprocessor Troubleshooting.
An Introduction to Z80 Microprocessor Troubleshooting.
68000 Microprocessor Concepts and Applications.
An Introduction to 68000 Microprocessor Applications.

Contents

Continued ...

Appendices		Pages

Chapter 9 Programs with Loops

Objectives of this Chapter

Having studied this chapter you will be able to:

- Describe the different types of program loop structure.
- Describe the use of the conditional and unconditional instructions.
- Explain the mechanism and use of Z80 relative addressing.
- Describe the function and operation of the following Z80 flags:
  - Carry Flag
  - Zero Flag
- Write programs that use the Z80 assembly language conditional and unconditional Jump instructions.
- Use the Z80 assembly language combined Decrement and conditional Jump instructions.

Equipment Required for this Chapter

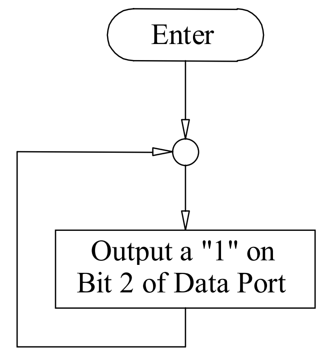
- SAM Z80 Microcomputer.
- Power supply.
- Keypad/display unit.
- Merlin Development System Software Pack, installed on a PC running Windows 95 or later.
- SAM Z80 User Manual.

Introduction

Often it will be necessary to use a program **loop** to **repeat** a section of a program a number of times. There are three main types of program loop:

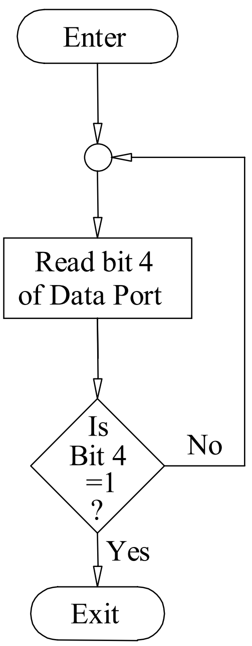
1. Repeating a program section indefinitely.

For example: Output a “1” on bit 2 of a data port indefinitely.



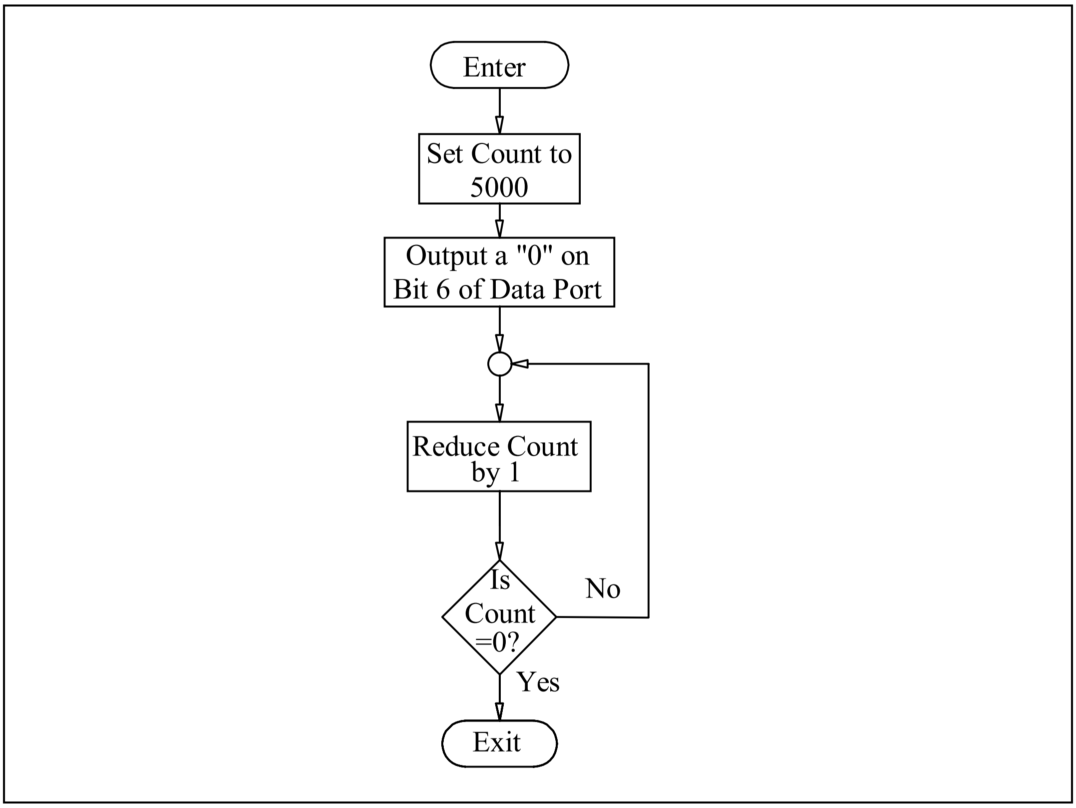
2. Repeating a program section until some predetermined condition becomes true.

For example: Waiting for a “1” to be input at bit 4 of a data port.



3. Repeating a program section for a predetermined number of passes.

For example: Output a “0” on bit 6 of a data port for the time it takes to repeat a loop 5000 times.



If, in this example, each pass through the loop were to take **1µs**, a "0" would be output on bit 6 of the data port for **5ms**.

In order to write assembly language programs with loops, it will be necessary to use JUMP instructions.

9.1 JUMP Instructions

These instructions cause program execution to be continued from some point other than the next location in sequence.

There are two types of JUMP instruction:

Unconditional JUMP	-	“Always JUMP”
Conditional JUMP	-	“Only JUMP if some condition is true”

9.2 Relative Addressing

Some JUMP instructions can use Relative Addressing. In this mode of addressing, the destination for the JUMP is not specified directly (for example, "location 4700_H") but is expressed in terms of the number of locations further on (or back) in the program (for example, "8 locations further on"). The easiest means of understanding this concept is to examine an example.

Recall the source program PROG2.ASM from Chapter 6. Load this program into Merlin and modify each "JP" instruction to "JR". A "JR" is a relative jump instruction. Having modified the program, save it as PROG4.ASM (using the **Save As** option from the **File** menu) and then assemble the program generating the object program and listing. The listing will have the form shown below:

```
                               ORG 4500H         ;Object code start addr

                       VAL1:   EQU 02H           ;Defines 'VAL1' as 02H
                       VAL2:   EQU 03H           ;Defines 'VAL2' as 03H
                       MEM1:   EQU 5000H         ;Defines 'MEM1' as 5000H

4500  3E 02            BEGIN:  LD A,VAL1         ;Loads accum with 02H
4502  18 04                    JR NEXT           ;Jumps to 'NEXT:'
4504  32 00 50         LAST:   LD (MEM1),A       ;Saves accum in 5000H
4507  C9                       RET               ;Return
4508  C6 03            NEXT:   ADD A,VAL2        ;Adds 03H to accum
450A  18 F8                    JR LAST           ;Jumps to 'LAST:'
```

Consider the first relative jump **`JR NEXT`**.

The operand is 04_H. This is often called the **displacement** or **offset** in relative addressing. The offset indicates that the destination is 04_H locations further on. It is important to note that the program counter will already contain the address of the next instruction (4504_H). If 04_H is added to this then the address of the destination can be found:

$$4504_H + 04_H = 4508_H$$

This is indeed the case above where `'NEXT'` is 4508_H.

Consider now the second relative jump **`JR LAST`**.

The offset is $F8_H$. This is a negative number in 2's complement notation.

$$\begin{aligned} F8_H &= +\ 1111\ 1000_2 \\ &= -\ 0000\ 0111_2 \text{ (1's complement)} \\ &= -\ 0000\ 1000_2 \text{ (2's complement)} \\ &= -\ 08_H \end{aligned}$$

So this instruction will jump backwards 08_H locations. Now, as before the program counter will already hold the address of the next instruction ($450C_H$ in this case):

$$450C_H - 08_H = 4504_H$$

So this instruction will jump to 4504_H. Check the program listing to see that this is indeed the case.

Range of Relative Addressing

The largest positive offset will be $7F_H$ ($0111\ 1111_2$), which is 127_{10}. So it is not possible to perform a relative jump more than 127_{10} locations in the forward direction. The largest negative offset will be 80_H ($1000\ 0000_2$).

$$\begin{aligned} 80_H &= +\ 1000\ 0000_2 \\ &= -\ 0111\ 1111_2 \text{ (1's complement)} \\ &= -\ 1000\ 0000_2 \text{ (2's complement)} \\ &= -\ 80_H \\ &= -\ 128_{10} \end{aligned}$$

So the limit of a backward relative jump is 128_{10} locations. The Z80 Cross Assembler will produce an error message if a relative jump is out of range.

9.2a **Z80 instructions, which allow programs to continue from a point other than the next location in sequence, are called:**

- [a] continue instructions.
- [b] jump instructions.
- [c] sequence instructions.
- [d] skip instructions.

9.2b **In relative addressing, the destination for a jump is specified by:**

- [a] a 2's complement displacement.
- [b] a direct address.
- [c] the contents of the flag register.
- [d] the contents of the BC register pair.

9.3 Conditional Jump Instructions

A conditional jump is only taken if some predetermined condition is true. Otherwise the next instruction in sequence is executed. These instructions are very important since they allow the microprocessor to take decisions. Conditional jumps may use direct or relative addressing. The conditions, which these instructions test, are the states of the flags.

Z80 Flags

Each flag is a single flip-flop, which can store a 0 or a 1. These flags indicate the type of result from the last ALU operation. Many instructions will affect various flags but a significant number do not (notably Load). The Z80 Programming Manual explains which flags are affected by each instruction. The Z80 has 6 flags but we shall only consider two for the time being: the Carry Flag and the Zero Flag:

1. Carry Flag

This flag is set (that is = 1) if the last arithmetic operation produced a “carry out”. For example:

If $3A_H$ is added to 47_H, the result is 81_H and there is no carry out:

$$\begin{array}{rl} 3A_H & \\ 47_H & + \\ \hline 81_H & \\ \hline \end{array} \qquad\qquad \begin{array}{rl} 0011\ 1010_2 & \\ 0100\ 0111_2 & + \\ \hline 1000\ 0001_2 & \\ \hline \end{array}$$

However, if $3A_H$ is added to $E7_H$, the result is 121_H. Thus a carry out is generated:

$$\begin{array}{rl} 3A_H & + \\ E7_H & \\ \hline 121_H & \\ \hline \end{array} \qquad 1 \qquad \begin{array}{rl} 0011\ 1010_2 & + \\ 1110\ 0111_2 & \\ \hline 1010\ 0001_2 & \\ \hline \end{array}$$

Carry out

Now, the accumulator can only hold 8 bits but this result is 9 bits in length. The 8 least significant bits (that is 21_H) will be placed in the accumulator and the “9th bit” in the carry flag, so, C=1.

The carry flag is also used as a “borrow” flag when performing subtraction.

2. Zero Flag

This flag is set (that is = 1) if the result of the last operation was zero.

For example, if the microprocessor subtracts 34_H from 34_H the result is 00_H and the zero flag is set (Z=1).

If 34_H is added to 34_H the result is 68_H which is non-zero so the zero flag is cleared (Z=0).

The action of these flags can be summarized thus:

Z	Zero Result	(Z=1)
NZ	Non-Zero Result	(Z=0)
C	Carry Generated	(C=1)
NC	No Carry Generated	(C=0)

Examples:

```
        JP Z,4820H        ;Jump to location 4820H if the
                          ;zero flag is set (that is if Z=1)
                          ;- Zero Result

        JR NZ,SCAN1       ;Jump to address specified by
                          ;the label SCAN1 if the zero
                          ;flag is clear(that is if Z=0)
                          ;- Non-Zero Result

        JR C,NEXT         ;Jump to address specified by
                          ;the label NEXT if the carry
                          ;flag is set(that is if C=1)
                          ;- Carry Generated

        JP NC,LAST        ;Jump to address specified by
                          ;the label LAST if the carry
                          ;flag is clear (that is if C=0)
                          ;- No Carry Generated
```

Now refer to your Z80 programming manual. Note how the Zero and Carry Flags are affected by each instruction:

Instructions	**Zero Flag**	**Carry Flag**
LD	Not affected	Not affected
ADD	Set if result is zero; otherwise cleared	Set if result exceeds 8 bits; otherwise cleared
SUB	Set if result is zero; otherwise cleared	Set if there is a ‘borrow’; otherwise cleared
RET	Not affected	Not affected
JP	Not affected	Not affected
INC	Set if result is zero; otherwise cleared	Not affected
DEC	Set if result is zero; otherwise cleared	Not affected

9.3a **After the Z80 has subtracted $4A_H$ from 67_H, the Zero (Z) and Carry (C) flags will be:**

- a Z = 0, C = 0
- b Z = 0, C = 1
- c Z = 1, C = 0
- d Z = 1, C = 1

9.3b **After the Z80 has added 52_H to 67_H, the Zero (Z) and Carry (C) flags will be:**

- a Z = 0, C = 0
- b Z = 0, C = 1
- c Z = 1, C = 0
- d Z = 1, C = 1

9.3c **After the Z80 has added 75_H to $8E_H$, the Zero (Z) and Carry (C) flags will be:**

- a Z = 0, C = 0
- b Z = 0, C = 1
- c Z = 1, C = 0
- d Z = 1, C = 1

9.3d **After the Z80 has subtracted 72_H from 72_H, the Zero (Z) and Carry (C) flags will be:**

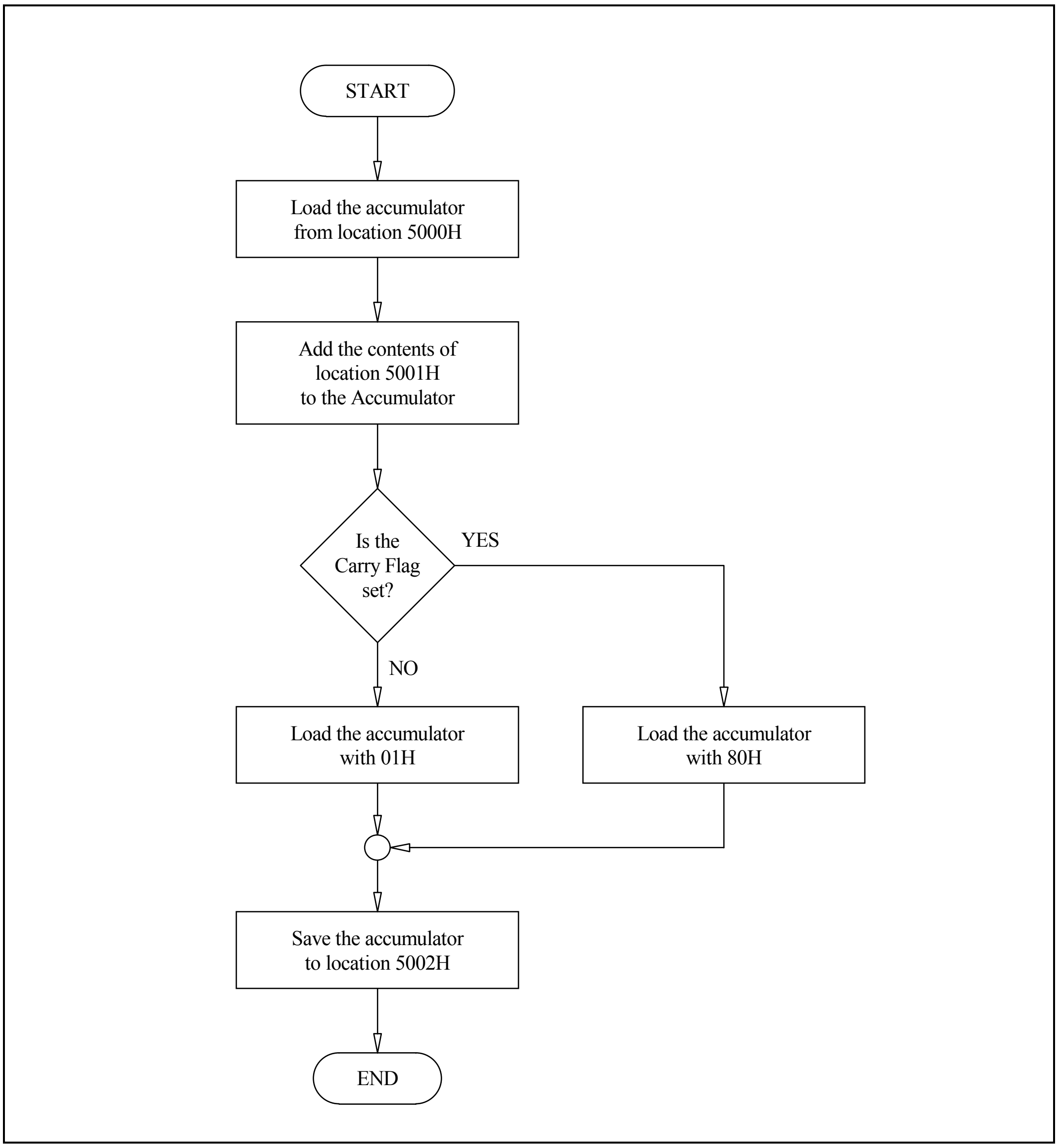
- a Z = 0, C = 0
- b Z = 0, C = 1
- c Z = 1, C = 0
- d Z = 1, C = 1

9.4 Worked Example

Write a program that will add the contents of locations 5000_H and 5001_H. The value 80_H should be placed in location 5002_H if the result exceeds FF_H, otherwise 01_H should be placed in location 5002_H.

Solution:

This problem requires the carry flag to be tested following the addition and then a marker value to be saved to indicate the status of the result.



The first part of this program will be quite simple:

```
                LD A,(5000H)     ;Loads accumulator from 5000H
```

Now, unfortunately direct addressing cannot be used with ADD. It is however possible to ADD the contents of two registers so one way to overcome the problem is to LOAD the B register from 5001_H and then ADD the B register to the accumulator.

A further problem now arises: It is not possible to load the B register using direct addressing. This problem can be overcome thus:

```
                LD A,(5000H)     ;Loads accumulator from 5000H
;Add contents of 5001H to accumulator:
                LD B,A           ;Loads the B reg from the accum
                LD A,(5001H)     ;Loads acc from 5001H
                ADD A,B          ;Adds B reg to accumulator
```

The addition has now taken place and so we can test the carry flag thus:

```
                LD A,(5000H)     ;Loads accumulator from 5000H
;Add contents of 5001H to accumulator:
                LD B,A           ;Loads the B reg from the accum
                LD A,(5001H)     ;Loads accumulator from 5001H
                ADD A,B          ;Adds B reg to accumulator
;Test for carry flag set:
                JR C,OVER        ;If C=1, jump to label OVER
```

The state of the carry flag will now determine which branch is taken:

```
                LD A,(5000H)     ;Loads accumulator from 5000H

;Add contents of 5001H to accumulator:
                LD B,A           ;Loads the B reg from the accum
                LD A,(5001H)     ;Loads accumulator from 5001H
                ADD A,B          ;Adds B reg to accumulator

;Test for carry flag set:

                JR C,OVER        ;If C=1, jump to label OVER

;Load accumulator with appropriate marker value:

                LD A,01H         ;C=0 so load marker for no carry
                JR SAVE          ;Jump to label SAVE(Save marker)
        OVER:   LD A,80H         ;Load accumulator with
                                 ;marker for carry set
```

All that is required now is to save the marker in location 5002_H:

```
                LD A,(5000H)     ;Loads accumulator from 5000H
;Add contents of 5001H to accumulator:
                LD B,A           ;Loads the B reg from the accum
                LD A,(5001H)     ;Loads accumulator from 5001H
                ADD A,B          ;Adds B reg to accumulator
;Test for carry flag set:
                JR C,OVER        ;If C=1, jump to label OVER
;Load accumulator with appropriate marker value:
                LD A,01H         ;C=0 so load marker for no carry
                JR SAVE          ;Jump to label SAVE (Save marker)
     OVER:      LD A,80H         ;Load accumulator with
                                 ;marker for carry set
;Save marker value in location 5002H:
     SAVE:      LD (5002H),A     ;Save marker in 5002H
                RET              ;Return
```

Finally, it will be necessary to specify the start address for object code using ORG:

```
                ORG 4400H        ;Start address for object code
                LD A,(5000H)     ;Loads accumulator from 5000H
;Add contents of 5001H to accumulator:
                LD B,A           ;Loads the B reg from the accum
                LD A,(5001H)     ;Loads accumulator from 5001H
                ADD A,B          ;Adds B reg to accumulator
;Test for carry flag set:
                JR C,OVER        ;If C=1, jump to label OVER
;Load accumulator with appropriate marker value:
                LD A,01H         ;C=0 so load marker for no carry
                JR SAVE          ;Jump to label SAVE (Save marker)
     OVER:      LD A,80H         ;Load accumulator with
                                 ;marker for carry set
;Save marker value in location 5002H:
     SAVE:      LD (5002H),A     ;Save marker in 5002H
                RET              ;Return
```

9.4a **Load the program for Worked Example 9.4 into the SAM. Place the value 67_H in location 5000_H and the value $7D_H$ in location 5001_H. Run the program and examine the contents of location 5002_H. Enter the hexadecimal byte that you find.**

Now, the first part of this program required a number of instructions to transfer data from one register to another. This could have been avoided by using the HL register pair to point to memory locations, since it is possible to ADD in this way. Such a program is shown below:

```
                ORG 4400H          ;Start address for object code

                LD HL,5000H        ;HL points to 5000H

;Add contents of 5001H to accumulator:

                LD A,(HL)          ;Loads the accumulator from 5000H
                INC HL             ;HL now points to location 5001H
                ADD A,(HL)         ;Adds contents of 5001H to accum

;Test for carry flag set:

                JR C,OVER          ;If C=1, jump to label OVER

;Load accumulator with appropriate marker value:

                LD A,01H           ;C=0 so load marker for no carry
                JR SAVE            ;Jump to label SAVE (Save
                                   ;marker)
     OVER:      LD A,80H           ;Load accumulator with marker
                                   ;for carry set.

;Save marker value in location 5002H:

     SAVE:      INC HL             ;HL now points to 5002H
                LD (HL),A          ;Saves marker in 5002H
                RET                ;Return
```

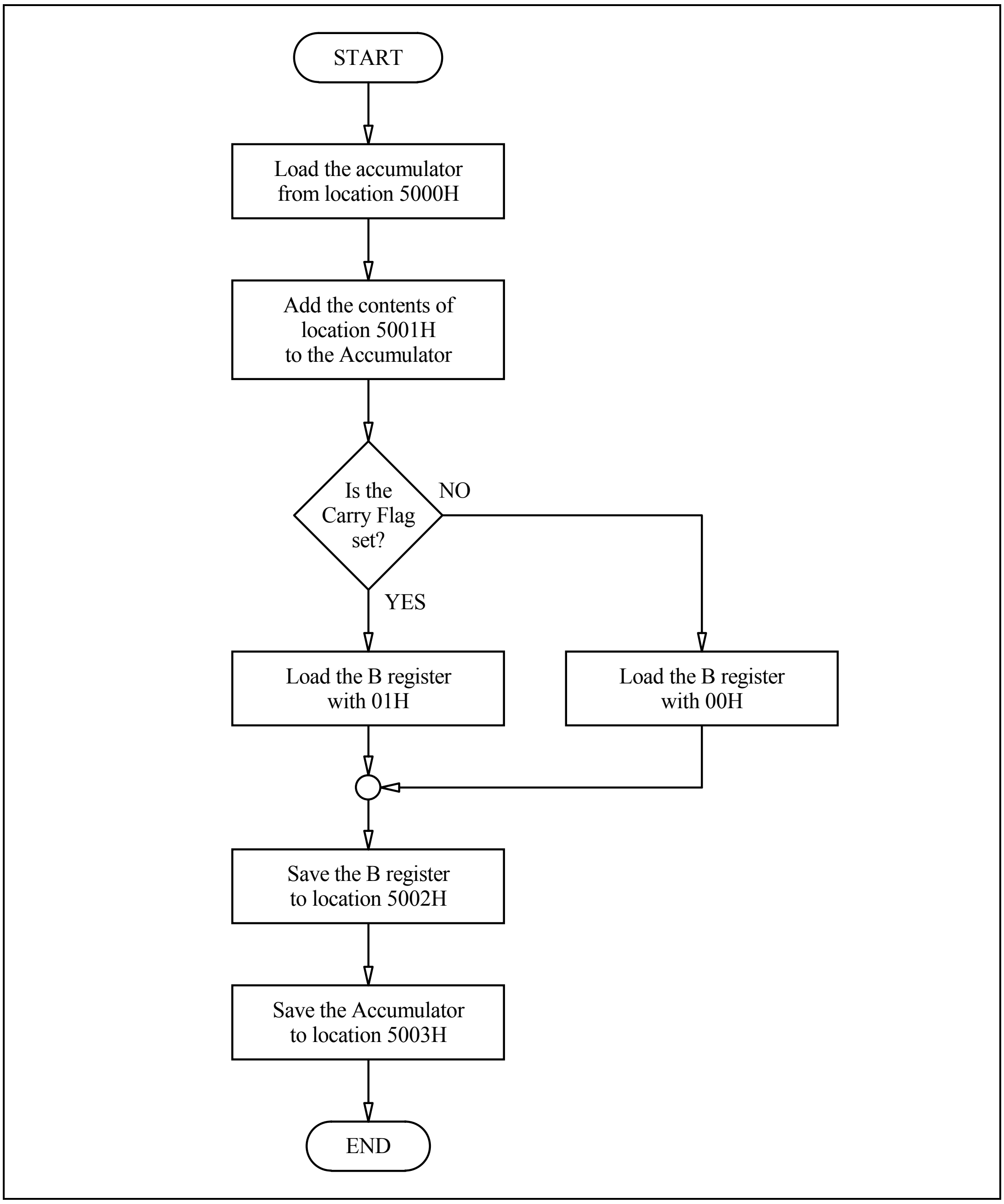
9.4b **Load the program for Worked Example 9.4 into the SAM. Place the value 82_H in location 5000_H and the value $9C_H$ in location 5001_H. Run the program and examine the contents of location 5002_H. Enter the hexadecimal byte that you find.**

Note: It is good practice to use the HL register pair to point to data wherever possible since this technique often takes fewer bytes of object code and less time to complete instructions.

9.5 Worked Example

Write a program that will add the contents of locations 5000_H and 5001_H. The most significant byte of the result should be stored in location 5002_H and the least significant byte in location 5003_H.

Solution:



Now, consider the largest possible values:

$$\mathbf{FF_H + FF_H = 01FE_H}$$

So the most significant byte can only be 00_H or 01_H.

Thus, the program must perform the addition, save the least significant byte and then test the carry flag to determine whether the most significant byte is 00_H or 01_H.

```
                ORG 4400H          ;Start address for object code

                LD HL,5000H        ;Loads accumulator from 5000H

;Add contents of 5001H to accumulator:

                LD A,(HL)          ;Loads the accumulator from 5000H
                INC HL             ;HL now points to location 5001H
                ADD A,(HL)         ;Adds contents of 5001H to accum

;Test for carry flag set:

                JR NC,ZERO         ;If C=0, jump to label ZERO

;Load accumulator with most significant byte:

                LD B,01H           ;C=1 so most significant byte is 01H
                JR SAVE            ;Jump to label SAVE (Save most
                                   ;significant byte)
        ZERO:   LD B,00H           ;C=0 so most significant byte is 00H

;Save most significant byte in location 5002H:

        SAVE:   INC HL             ;HL now points to 5002H
                LD (HL),B          ;Saves most significant byte in 5002H
                INC HL             ;HL now points to 5003H
                LD (HL),A          ;Saves least significant byte in
                                   ;5003H
                RET                ;Return
```

9.5a **Load the program for Worked Example 9.5 into the SAM. Place the value $3E_H$ in location 5000_H and the value $E5_H$ in location 5001_H. Run the program and examine the contents of locations 5002_H and 5003_H. Enter the 2-byte hexadecimal result that these locations represent.**

9.6 Practical Assignment

Write a program that will examine the contents of location 5000_H. If the contents are 00_H, location $5FFF_H$ should be loaded with 80_H. If the contents are non-zero then $5FFF_H$ should be loaded with $7F_H$.

[Hint: LOAD does not condition the flags. Use the instruction "`OR 00H`" to condition the flags prior to testing.]

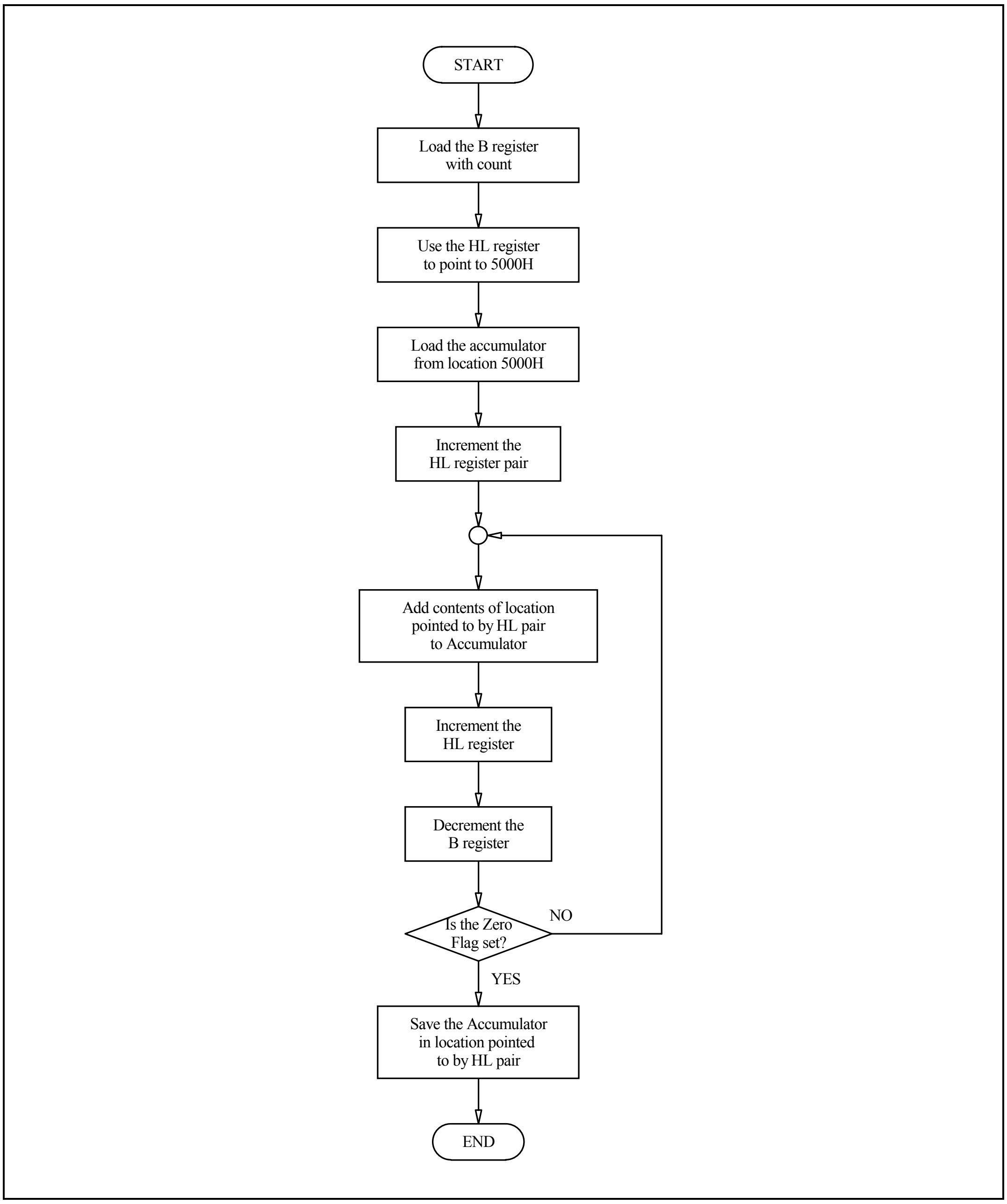
9.6a **Load the program for Practical Assignment 9.6 into the SAM. Place the value $2A_H$ in location 5000_H. Run the program and examine the contents of location $5FFF_H$. Enter the hexadecimal byte that you find.**

So far the programs you have entered have been decision-making rather than loops. Consider now the problem of repeating a section of a program a given number of times. These types of programs often use a register or memory location as a **loop counter**. The loop counter is decremented (decreased by 01_H) on each pass through the loop and tested for zero. When the counter reaches zero the program exits from the loop and continues.

9.7 Worked Example

Write a program, which will add together, the contents of locations 5000_H, 5001_H, 5002_H, 5003_H and 5004_H, saving the result in location 5005_H.

Solution:



A source program for this is shown below. Note that the B-register can be used as a convenient loop counter:

```
           ORG 4400H          ;Start address for object code
           LD B,04H           ;Load the B register with count
           LD HL,5000H        ;HL register points to location 5000H
           LD A,(HL)          ;Loads the accumulator from location
                              ;5000H
           INC HL             ;Increase the HL register by 01H
NEXT:      ADD A,(HL)         ;Adds contents of location
                              ;pointed to by HL register pair to
                              ;the accumulator
           INC HL             ;Increase the HL register by 01H
           DEC B              ;Decrease the count by 01H
           JR NZ,NEXT         ;Jump to the label NEXT if the zero
                              ;flag is NOT SET (that is Z=0). This
                              ;indicates that count has not yet
                              ;reached zero
           LD (HL),A          ;Save accumulator in location pointed
                              ;to by HL pair
           RET                ;Return
```

9.7a **If the program in Worked Example 9.7 is to be modified to add the contents of locations 5000_H, 5001_H and 5002_H, saving the result in location 5003_H, the instruction that must be changed is:**

- a `LD B,04H`
- b `LD HL,5000H`
- c `LD A,(HL)`
- d `DEC B`

Now, the Z80 has a special instruction which combines decrementing of the B register with testing whether the result is zero - DJNZ (Decrement the B register and jump if result is non-zero). The previous program could be easily modified to take advantage of this instruction thus:

```
                ORG 4400H          ;Start address for object code
                LD B,04H           ;Load the B register with count
                LD HL,5000H        ;HL register points to location 5000H
                LD A,(HL)          ;Loads the accumulator from
                                   ;location 5000H
                INC HL             ;Increase the HL register by 01H
     NEXT:      ADD A,(HL)         ;Adds contents of pointed to by HL
                                   ;register
                                   ;pair to the accumulator
                INC HL             ;Increase the HL register by 01H
                DJNZ NEXT          ;Decrement the B register and
                                   ;jump to the
                                   ;label NEXT if the result is non-zero
                LD (HL),A          ;Save accumulator in location
                                   ;pointed to by HL pair
                RET                ;Return
```

9.7b **Place the values shown below in the memory locations indicated.**

Memory locations	**Contents**
5000_H	**12_H**
5001_H	**$0C_H$**
5002_H	**39_H**
5003_H	**$0F_H$**
5004_H	**$2B_H$**

Load the modified program for Worked Example 9.7 into the SAM and run. Enter the hexadecimal byte that you find in location 5005_H.

9.8 Practical Assignment

A simple means of achieving multiplication is to add a value to itself a given number of times. Location 5000_H contains a value between 00_H and 33_H. Use this method to multiply the contents of location 5000_H by 05_H, saving the result in location 5001_H.

9.8a **Use your program for Practical Assignment 9.8 to calculate 28_H x 05_H. Enter the result that you find.**

9.8b **Modify your program for Practical Assignment 9.8 to calculate $1E_H$ x 07_H. Enter the result that you find.**

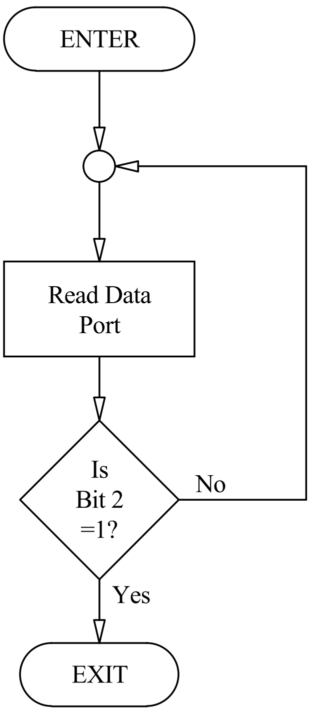
Student Assessment 9

1. The type of structure used to repeat a section of program several times is called:

- a an Echo.
- b a Go To.
- c a Loop.
- d a Repeat.

2. The program section described by the flowchart below will:

- a repeat indefinitely.
- b repeat until a condition becomes true.
- c repeat for a given number of passes.
- d not repeat.



Student Assessment 9 Continued ...

3. The type of JUMP that is always taken is called a:

- a Conditional Jump.
- b Direct Jump.
- c Indirect Jump.
- d Unconditional Jump.

4. The type of JUMP that allows the microprocessor to make decisions is called a:

- a Conditional Jump.
- b Direct Jump.
- c Indirect Jump.
- d Unconditional Jump.

5. The type of addressing where the destination is expressed in terms of the number of bytes forward or backward from the present location is called:

- a Conditional.
- b Direct.
- c Indirect.
- d Relative.

6. The largest *positive* 8-bit offset for relative addressing is:

- a 125_{10}
- b 126_{10}
- c 127_{10}
- d 128_{10}

Continued ...

Student Assessment 9 Continued ...

7. **The assembly language instruction at location 4418_H is `'JR NZ,WRIPT'`. If the location identified by the label `'WRIPT'` is $441E_H$, the 2's complement displacement for the branch instruction will be:**

   a $F8_H$

   b 04_H

   c FA_H

   d 06_H

8. **The Carry Flag is set when the result of the last arithmetic operation is:**

   a zero.

   b non-zero.

   c less than 8 bits.

   d greater than 8 bits.

9. **The Flag that is set when the result of the last arithmetic operation is zero is the:**

   a Carry Flag.

   b Negative Flag.

   c Overflow Flag.

   d Zero Flag.

Student Assessment 9 Continued ...

10. The program section, which will repeatedly (and indefinitely) add 02_H to the Accumulator, is:

a
```
HERE:   ADD A,02H
        JR HERE
```
b
```
HERE:   ADD A,02H
        JR NC,HERE
```
c
```
HERE:   ADD A,02H
        JR C,HERE
```
d
```
HERE:   ADD A,02H
        JR NZ,HERE
```

11. The program section below

```
NEXT:   ADD A,B
        JR C,DONE
        JR NEXT
```

will add the contents of the B-Register to the Accumulator:

a indefinitely.

b until the result is greater than 8 bits.

c until the result is less than 8 bits.

d until the result is equal to contents of the B-Register.

12. The Z80 instruction, which decrements the B-Register and tests whether the result is zero, is:

a `DAA`

b `DEC`

c `DI`

d `DJNZ`