EEEN 3449
Microprocessor Systems

Bit Testing

Tyler Hurson

Spring 2017

I. INTRODUCTION

1.1 Purpose

The purpose of this experiment is explore the AND, OR, and XOR operations and how they affect the bits in particular numbers using the Assembly language.

1.2 Problem

Often, individual bits must be set, cleared, or toggled. This can be achieved by using the AND, OR, and XOR operations. AND has the effect of clearing bits; the 0 bits in the operand indicate which bits will be cleared. OR has the effect of setting bits; the 1 bits in the operand indicate which bits will be set. XOR has the effect of toggling bits; the 1 bits in the operand indicate which bits will be toggled.

Five variations of a program were created. Initially, $5A was loaded to accumulator A from memory address $1500. A set of 1-2 bitwise operations were performed, and then the number was stored back into $1500. Program A (Appendix A) toggled bits 0-3 and cleared bits 4-7. Program B (Appendix B) toggled bits 4-7. Program C (Appendix C) toggled bits 4-7 and set bits 0-3. Program D (Appendix D) toggled bits 0-3 and set bits 4-7. Program E (Appendix E) set bits 7, 5, 3, 1 and cleared bits 6, 4, 2, 0.

Two additional variation of a program were created. Program F (Appendix) loaded $5A into accumulator B. B was then rotated 4 times to the right, effectively swapping the two nibbles in the byte. The number was then stored back into $1500. Program G (Appendix) is similar to program F, but used accumulator A instead of accumulator B to store $5A.

1.3 Scope

The scope of this experiment is limited to the HCS12 microcontroller. Several instructions will be used from the HCS12 instruction set.

II. TEST AND EVALUATION

2.1 Apparatus

The equipment used in this test includes: Dragon12-Junior development board, USB power cord, and laptop PC with AsmIDE.

2.2 Procedure

1. The development board was connected to the computer.
2. The COM port number was determined under Device Manager on PC. AsmIDE was launched. Under View -> Options -> COM Port, the COM port was set to the device's number. The Terminal Window was enabled. Under Set COM Options, the default values were restored.
3. Program A was opened, and then assembled. After no errors were recorded, program A was downloaded into the development board, by typing `load` in the Terminal Window in AsmIDE, then downloading the program.
4. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.
5. Program B was opened, and then assembled. After no errors were recorded, program B was downloaded into the development board.
6. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.
7. Program C was opened, and then assembled. After no errors were recorded, program C was downloaded into the development board.

8. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.
9. Program D was opened, and then assembled. After no errors were recorded, program D was downloaded into the development board.
10. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.
11. Program E was opened, and then assembled. After no errors were recorded, program E was downloaded into the development board.
12. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.
13. Program F was opened, and then assembled. After no errors were recorded, program F was downloaded into the development board.
14. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.
15. Program G was opened, and then assembled. After no errors were recorded, program G was downloaded into the development board.
16. `g 2000` was typed to execute the program. At the end of the program, `md 1500` was typed to confirm that the number stored at $1500 was correct.

III. RESULTS

3.1 Data

Table 1 displays the final result (the number stored at $1500) of each of the seven programs.

Table 1: Final Results

Program	Result
A	$05
B	$AA
C	$AF
D	$F5
E	$66
F	$A5
G	$A5

3.2 Analysis

The following flowcharts display the program flow for each of the seven programs executed.

Figure 2: Flowchart of Program A

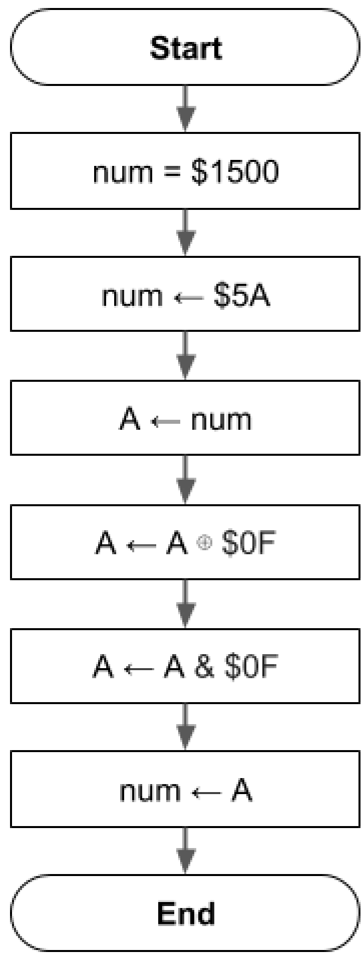


Figure 3: Flowchart of Program B

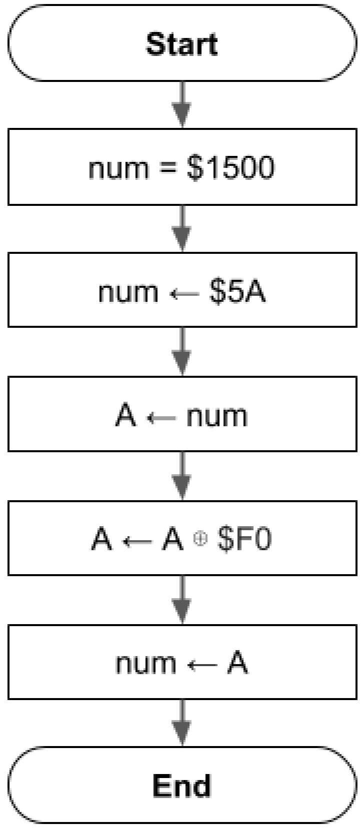


Figure 4: Flowchart of Program C

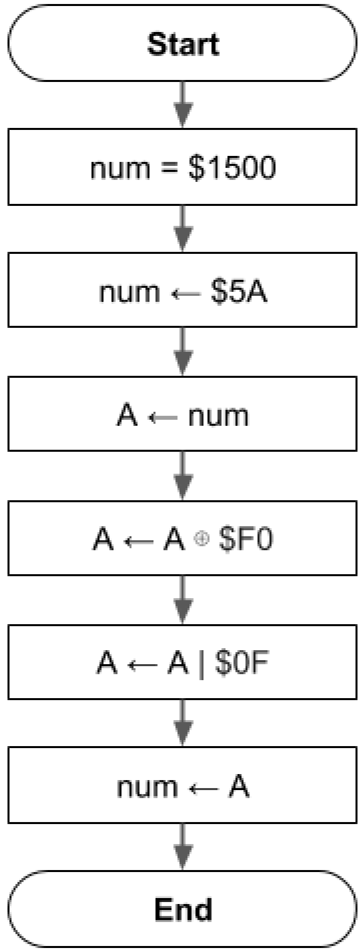


Figure 5: Flowchart of Program D

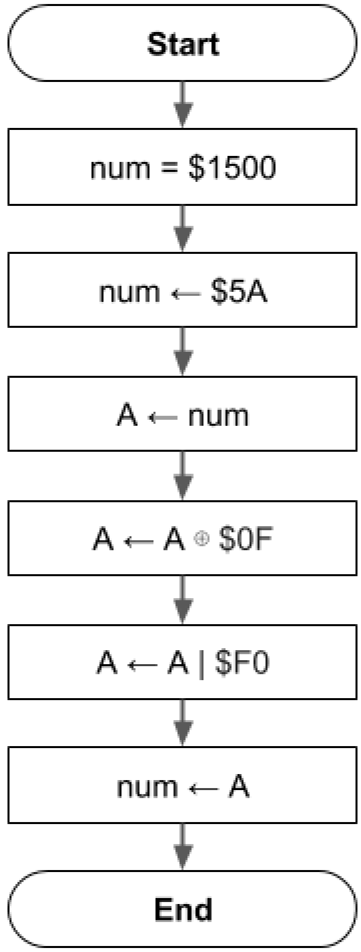


Figure 6: Flowchart of Program E

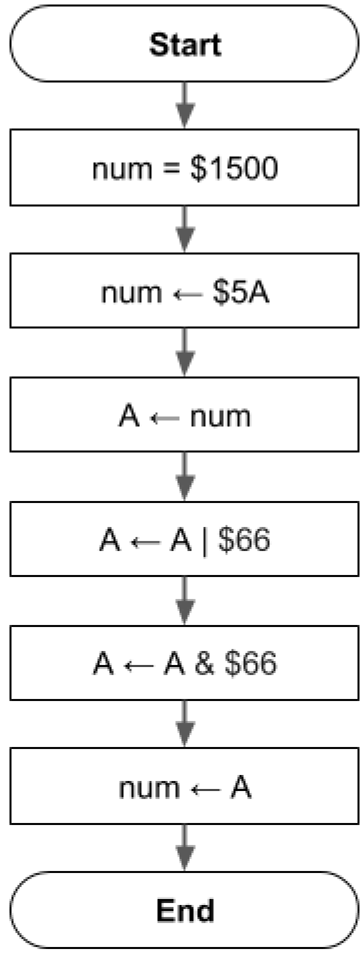


Figure 7: Flowchart of Program F

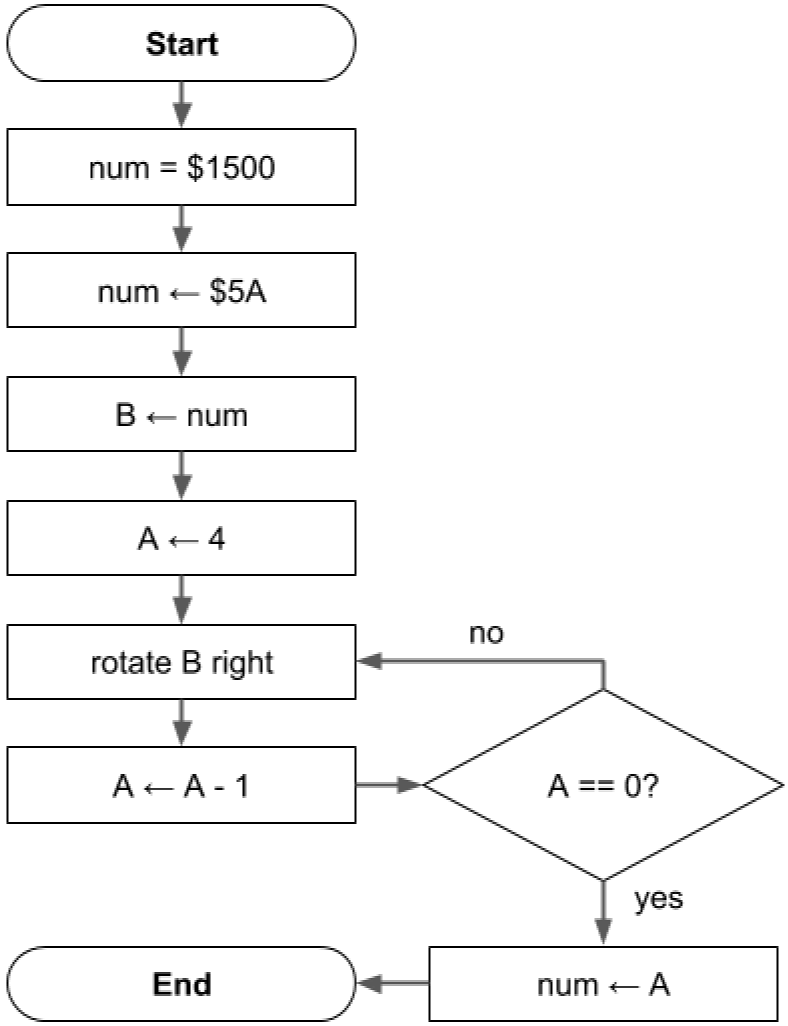
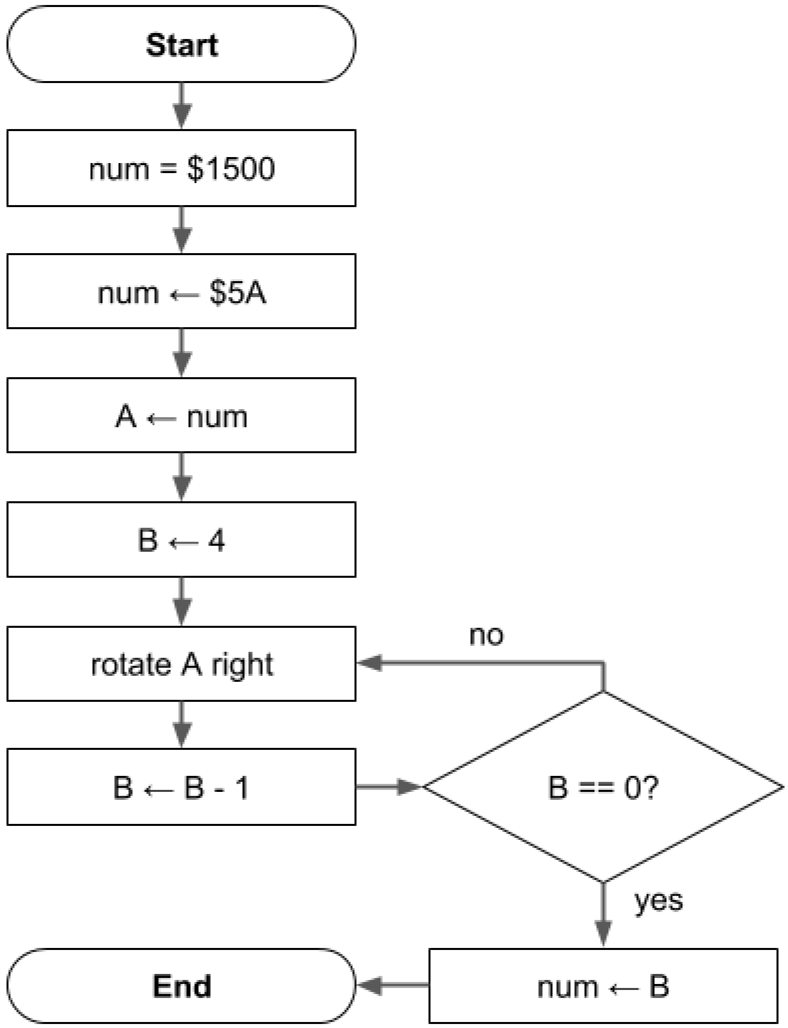


Figure 8: Flowchart of Program G



III. CONCLUSION

4.1 Assessment

This experiment served as an introduction to bit setting/clearing/toggling. Bitwise manipulation is useful when working with boolean values, which may be represented as a single flag (bit) in a 8-bit value.

APPENDIX A

ASSEMBLY PROGRAM A

```
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num        ; load $5A into $1500
        ldaa    num             ; A = $5A
        eora    #$0f            ; toggle bits 3,2,1,0 in A
        anda    #$0f            ; clear bits 7,6,5,4 in A
        staa    num             ; $1500 = A
        swi
        end
```

APPENDIX B

ASSEMBLY PROGRAM B

```
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num        ; load $5A into $1500
        ldaa    num             ; A = $5A
        eora    #$f0            ; toggle bits 7,6,5,4 in A
        staa    num             ; $1500 = A
        swi
        end
```

APPENDIX C

ASSEMBLY PROGRAM C

```
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num        ; load $5A into $1500
        ldaa    num             ; A = $5A
        eora    #$f0            ; toggle bits 7,6,5,4 in A
        oraa    #$0f            ; set bits 0,1,2,3 in A
        staa    num             ; $1500 = A
        swi
        end
```

APPENDIX D

ASSEMBLY PROGRAM D

```
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num         ; load $5A into $1500
        ldaa    num              ; A = $5A
        eora    #$0f             ; toggle bits 0,1,2,3 in A
        oraa    #$f0             ; set bits 7,6,5,4 in A
        staa    num              ; $1500 = A
        swi
        end
```

APPENDIX E

ASSEMBLY PROGRAM E

```
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num        ; load $5A into $1500
        ldaa    num             ; A = $5A
        oraa    #$66            ; set bits 7,5,3,1 in A
        anda    #$66            ; clear bits 6,4,2,0 in A
        staa    num             ; $1500 = A
        swi
        end
```

APPENDIX F

ASSEMBLY PROGRAM F

```
N       equ     4
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num        ; load $5A into $1500
        ldab    num             ; B = $5A
        ldaa    #N              ; A = 4
loop    rorb                    ; rotate B to right
        dbne    A,loop          ; decrement A, branch if A != 0
        stab    num             ; $1500 = B
        swi
        end
```

APPENDIX G

ASSEMBLY PROGRAM G

```
N       equ     4
        org     $1500
num     rmb     1
        org     $2000
        movb    #$5A,num        ; load $5A into $1500
        ldaa    num             ; A = $5A
        ldab    #N              ; B = 4
loop    rora                    ; rotate A to right
        dbne    B,loop          ; decrement B, branch if B != 0
        staa    num             ; $1500 = A
        swi
        end
```