CPE 325: Intro to Embedded Computer System

Lab04

MSP430 Assembler Directives and Addressing Modes

Submitted by: Gianna Foti

Date of Experiment:_____16 September 2024_____

Report Deadline:_____17 September 2024_____

Lab Section: _____CPE353-02_____

Demonstration Deadline: _____17 September 2024_____

Introduction

In this lab, we had to develop two assembly programs for the MSP430 microcontroller. Additionally, I wrote a bonus program. First, I wrote a program to analyze a string by counting the number of digits and the total length. I set Port 1 (P1) and Port 2 (P2) to display these counts. For the string “Welcome 2 the MSP430 Assembly!”, the program correctly calculated the digits and the length of the characters, outputting to P1OUT and P2OUT. Next, we had to create a program to determine if a given integer was odd or even. The result was stored in memory. For the bonus task, I converted all uppercase letters in the string “WELCoME To MSP430 ASSEMBLY!” to lowercase and count the changes. The updated string was “welcome to msp430 assembly!” and the total changes were shown on P3OUT. These tasks involved manipulating strings and integers using assembly language, showcasing skills in bitwise operations and memory management on the MSP430 platform.

Theory Topics

1. Assembler directives
    a. Assembler directives are instructions used in assembly language programming to control the assembly process. They do not generate machine code but provide information to the assembler about how to process the assembly code.
    b. Examples:
        i. .data: Defines a data segment where variables and constants are stored.
        ii. .text: Marks the beginning of the code segment containing executable instructions.
        iii. .end: Indicates the end of the source code to the assembler.
        iv. .byte, .word, .long: Allocate storage for variables in bytes, words, or long words, respectively.
        v. .space: Reserves a specified amount of space in memory without initializing it.

vi. .def: Defines symbols or constants in assembly language.

2. Addressing modes:
    a. Register: The operand is stored directly in a register. This is the fastest mode since it doesn't involve memory access.
    b. Indexed: Combines a base address (in a register) with an offset to point to the operand's memory location.
    c. Symbolic: Uses a symbol (label) as a memory address. The operand is at a memory address relative to the program counter.
    d. Absolute: Uses a fixed memory address directly in the instruction.
    e. Indirect: The register contains the address of the operand. The operand is in memory at the address pointed to by a register.
    f. Immediate: The operand is provided as a literal constant in the instruction.
    g. Indirect with autoincrement: The register points to the operand's memory address, and the register is automatically incremented after the operand is accessed.

Results & Observation

Part 1:

Program Description:

This program counts the number of digits and total characters in a predefined string and outputs these counts to Port 1 and Port 2, respectively. It processes the string character by character and updates the count.

Process:

1. Initialize System: Set up the stack pointer and stop the watchdog timer to prepare the microcontroller for execution.
2. Configure Ports: Set all pins on Port 1 (P1DIR) and Port 2 (P2DIR) to output mode.

3. Load String Address: Load the address of the string "Welcome 2 the MSP430 Assembly!" into register R5.
4. Initialize Counters: Clear registers R6 (for digit count) and R7 (for character count).
5. Count Characters and Digits:
    a. Loop through each character of the string:
        i. Check for End of String: If the current character is the NULL terminator (0), exit the loop.
        ii. Count Characters: Increment the character count (R7) for each character.
        iii. Check for Digits: If the character is between '0' and '9', increment the digit count (R6).
6. Output Results:
    a. Write the total number of characters to Port 2 (P2OUT).
    b. Write the number of digits to Port 1 (P1OUT).
7. Enter Low Power Mode:
    a. Transition the microcontroller to Low Power Mode 4 (LPM4) to save power.
    b. The nop instruction is used as a placeholder, typically for debugging.

Program 1 Output:

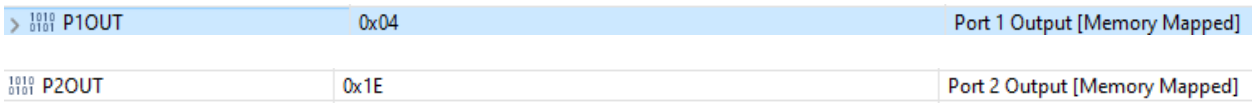

Figure 1: Program 1

Flowchart:

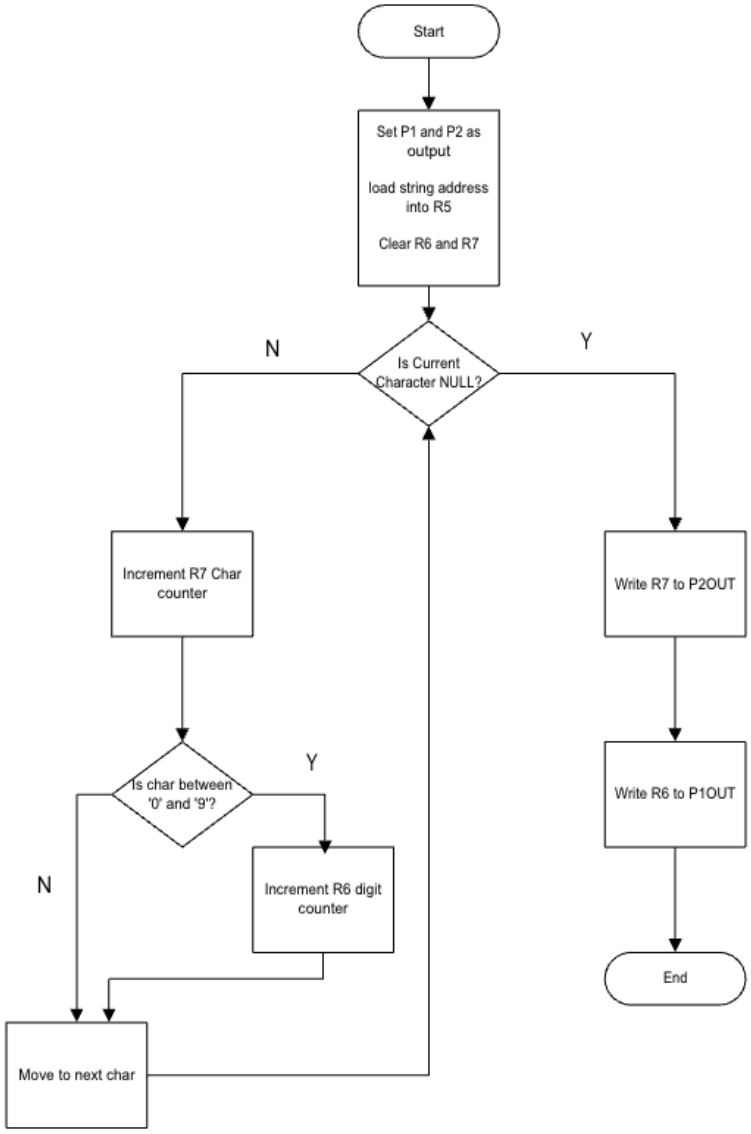

Figure 2: Program 1 Flowchart

Report Questions:

No Questions for Program 1.

Part 2:

Program Description:

This assembly program for the MSP430 microcontroller determines whether an integer value is even or odd and then stores the result as an ASCII string in memory.

Process:

1. Initialize System: Set up the stack pointer and stop the watchdog timer to prepare the microcontroller for operation.
2. Configure Port 1: Set all pins on Port 1 to output mode, although this is not used further in the code.
3. Load Integer: Retrieve the integer value 21 from TACO_LOVER and load it into register R5.
4. Check Even/Odd: Perform a bitwise AND operation with 1 to isolate the least significant bit (LSB) of the integer. This determines if the number is even or odd:
    a. Even: LSB is 0
    b. Odd: LSB is 1
5. Store Result:
    a. If Odd: Store the ASCII string "Odd" in RESULT_GHF.
    b. If Even: Store the ASCII string "Even" in RESULT_GHF.
6. Enter Low Power Mode: Transition the microcontroller into Low Power Mode 4 to save power and halt further CPU activity.
7. End Program: The program ends, with the result ("Even" or "Odd") stored in memory and the system in a low-power state.

Program 2 Output:

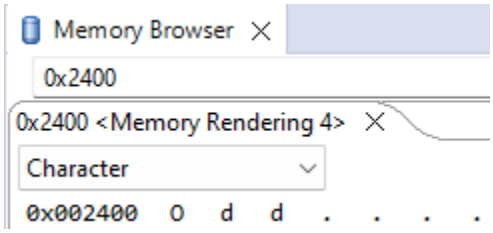

Figure 3: Program 2 Output with input 21

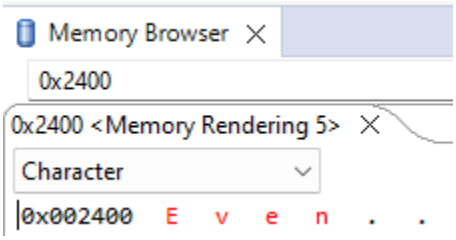

Figure 4: Program 2 Output with input 20

Flowchart:

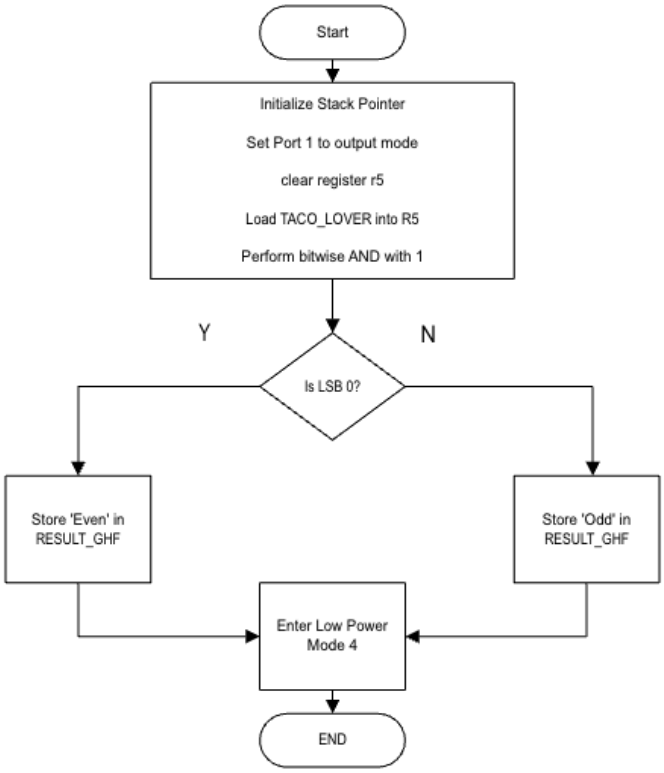

Figure 5: Program 2 Flowchart

Report Questions:

No Questions for Program 2.

Part 3:

BONUS Description:

This assembly program processes a string to convert all uppercase letters to their corresponding lowercase counterparts and counts the number of changes made. For the given input string "WELCoME To MSP430 ASSEMBLY!", the program will update it to "welcome to msp430 assembly!" and output the total number of changes (18 but 12 in hex) to P3OUT.

Process:

1. Initialize System: Set up the stack pointer and stop the watchdog timer to prepare the microcontroller for operation.
2. Configure Ports: Set Port 1 and Port 3 to output mode to use them for future operations and output results.
3. Load Addresses and Clear Counters: Load the address of the input string into register R4 and the address of the results buffer into register R8. Clear registers R5 (for character processing) and R6 (for counting changes), and initialize register R7 for counting string length (not necessary but included).
4. Process String:
    a. Loop Through Characters: Increment the length counter and load each character from the input string.
    b. Check for End of String: If the null terminator (0x00) is encountered, jump to the end of the program.
    c. Convert Uppercase to Lowercase: If the character is between 'A' and 'Z', convert it to lowercase by adding 32 to its ASCII value and increment the change counter.

d. Store Updated Character: Save the processed character into the results buffer and move to the next position.

5. Complete Processing: Continue processing characters until the end of the string is reached.
6. Output Results: After processing the entire string, write the total number of changes made (stored in register R6) to Port 3 (P3OUT).

Bonus Output:

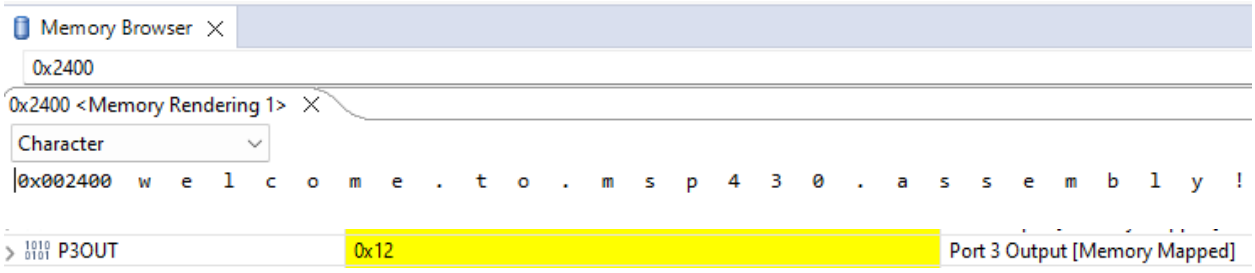

Figure 6: Bonus Output

Conclusion

In conclusion, I developed two assembly programs for the MSP430 microcontroller, each focusing on different aspects of embedded systems programming. In the first project, I created a checker that determines whether an integer is even or odd and stores the result as an ASCII string in memory. I also implemented efficient power management by putting the microcontroller into Low Power Mode 4. For the second project, I wrote a string analyzer that counts the number of digits and total characters in a string and outputs these counts to specific I/O ports. This project also involved transitioning to Low Power Mode 4 to save energy. I also did a bonus project as described above. Through these tasks, I gained hands-on experience with bitwise operations, character processing, and memory management. I did not have too many struggles and learned a lot through this lab. The hardest part was just learning the syntax of the MSP430 assembly language. In all, this was a great lab.

Appendix

Table 1: Program 1 source code

```
; -------------------------------------------------------------------------------
; File: Lab04_P1.asm
; Description: Counts the number of digits and total characters in a string
; Input: String
; Output: Number of digits in P1OUT | Number of characters in P2OUT
; Author: Gianna Foti
; -------------------------------------------------------------------------------
   .cdecls C, LIST, "msp430.h"     ; Include device header file
;-------------------------------------------------------------------------------
   .def    RESET                   ; Export program entry-point to linker
;-------------------------------------------------------------------------------
STRING:  .string "Welcome 2 the MSP430 Assembly!", "  ; Input string
;-------------------------------------------------------------------------------
   .text                           ; Assemble into program memory
RESET:
   mov.w   #__STACK_END, SP        ; Initialize stack pointer
   mov.w   #WDTPW|WDTHOLD, &WDTCTL ; Stop watchdog timer
;-------------------------------------------------------------------------------
; MAIN PROGRAM
;-------------------------------------------------------------------------------
   bis.b   #0FFh, &P1DIR           ; Set all P1 pins to output
   bis.b   #0FFh, &P2DIR           ; Set all P2 pins to output
   mov.w   #STRING, R5                     ; Load address of string into R5
   clr.b   R6                    ; Clear R6 (digit counter)
   clr.b   R7                    ; Clear R7 (character counter)

LOOP_GHF:
   mov.b   @R5, R8                 ; Load current character into R8
   cmp.b   #0, R8                  ; Check if it's the NULL terminator
   jeq     DONE                    ; If NULL, end loop
   inc.b   R7                    ; Increment character count
   cmp.b   #'0', R8                ; Compare with ASCII '0'
   jl      NEXT_CHAR               ; If less than '0', go to next character
   cmp.b   #'9', R8                ; Compare with ASCII '9'
   jge     NEXT_CHAR               ; If greater than or equal to '9', go to next character
   inc.b   R6                    ; If it's a digit, increment digit count

NEXT_CHAR:
   inc.w   R5                    ; Move to the next character in the string
   jmp     LOOP_GHF                ; Repeat the loop

DONE:
   mov.b   R7, &P2OUT              ; Write character count to P2OUT
   mov.b   R6, &P1OUT              ; Write digit count to P1OUT
   bis.w   #LPM4, SR               ; Enter low-power mode
   nop                           ; Placeholder for debugger

;-------------------------------------------------------------------------------
; Stack Pointer definition
;-------------------------------------------------------------------------------
   .global __STACK_END
```

```
   .sect   .stack

;-------------------------------------------------------------------------------
; Reset Vector
;-------------------------------------------------------------------------------
   .sect   ".reset"              ; MSP430 RESET Vector
   .short  RESET
   .end
```

Table 2: Program 2 source code

```
; -------------------------------------------------------------------------------
; File: Lab04_P2.asm
; Description: This program determines if an integer value is even or odd.
;              It outputs the ASCII string "Even" or "Odd" based on the value of TACO_LOVER.
; Input: An integer stored in TACO_LOVER
; Output: ASCII "Even" or "Odd" stored in memory at RESULT_GHF
; Author(s): Gianna Foti
; -------------------------------------------------------------------------------

      .cdecls C, LIST, "msp430.h"     ; Include device header file

;-------------------------------------------------------------------------------
      .def    RESET                 ; Export program entry-point to make it known to linker.

;-------------------------------------------------------------------------------
TACO_LOVER:  .int 21                ; Initialize TACO_LOVER with the value 20 or 21 for even or odd
                  .data
RESULT_GHF:     .space 30             ; Allocate at least 4 bytes of memory for the output string (only need 4)

;-------------------------------------------------------------------------------
      .text                         ; Assemble into program memory.
      .retain                        ; Override ELF conditional linking
                                    ; and retain current section.
      .retainrefs                    ; Retain any sections that have references to current section.

;-------------------------------------------------------------------------------
RESET:  mov.w   #__STACK_END,SP         ; Initialize the stack pointer
      mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop the watchdog timer

;-------------------------------------------------------------------------------
; Main Programmmm
;-------------------------------------------------------------------------------
main:
      bis.b   #0FFh, &P1DIR          ; Set Port 1 to output mode
      clr.b   R5                     ; Clear register R5

      mov.w   TACO_LOVER, R5         ; Load TACO_LOVER into R5
      and.w   #1, R5                 ; Perform bitwise AND with 1 to check the least significant bit

      jz      EVEN                   ; If LSB is 0 (even number), jump to EVEN
      jnz     ODD                    ; If LSB is 1 (odd number), jump to ODD

; Store the characters for "Odd" in memory allocated at RESULT_GHF
ODD:
                        ; O in hex
      mov.b   #0x4F, RESULT_GHF
      ; d in hex
```

```
        mov.b   #0x64, RESULT_GHF+1
        ; d in hex
        mov.b   #0x64, RESULT_GHF+2
        ; Null terminator for the string
        mov.b   #0x0, RESULT_GHF+3
        ; Jump to DAEND
        jmp     DAEND

; Store the characters for "Even" in memory allocated at RESULT_GHF
EVEN:

        mov.b   #0x45, RESULT_GHF
        mov.b   #0x76, RESULT_GHF+1
        mov.b   #0x65, RESULT_GHF+2
        mov.b   #0x6E, RESULT_GHF+3
        ; Jump to daend
        jmp     DAEND



DAEND:
                        ; Enter Low Power Mode 4
        bis.w #LPM4, SR
        ; No operation (typically used for debugging)
        nop

;-------------------------------------------------------------------------------
; Stack Pointer Definition
;-------------------------------------------------------------------------------
        .global __STACK_END
        .sect   .stack

;-------------------------------------------------------------------------------
; Interrupt Vectors
;-------------------------------------------------------------------------------
        .sect   ".reset"                ; MSP430 RESET Vector
        .short  RESET
        .end
```

Table 3: BONUS source code

```
; -------------------------------------------------------------------------------
; File: Lab04_Bonus.asm
; Description: Converts uppercase letters to lowercase and counts the changes.
; Input: String
; Output: Updated string and number of changes in P3OUT
; Author(s): Gianna Foti
; -------------------------------------------------------------------------------
        .cdecls C, LIST, "msp430.h"     ; Include device header file
;-------------------------------------------------------------------------------
        .def    RESET                   ; Export program entry-point to
                                        ; make it known to linker.
;-------------------------------------------------------------------------------
STRING:  .string "WELCoME To MSP430 ASSEMBLY!", "
        .data
RESULTS:     .space 64                  ; Allocate 64 bytes for outcome string
;-------------------------------------------------------------------------------
                                                                        ; built in crap
```

```
      .text                       ; Assemble into program memory.
      .retain                     ; Override ELF conditional linking
                                  ; and retain current section.
      .retainrefs                 ; And retain any sections that have
                                  ; references to current section.
;-------------------------------------------------------------------------------
RESET:  mov.w   #__STACK_END,SP         ; Initialize stack pointer
      mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; stop watchdog timer
;-------------------------------------------------------------------------------
; MAIN LOOP HERE (START OF PROGRAM)
;-------------------------------------------------------------------------------
main:   bis.b   #0FFh, &P1DIR           ; set P1
      bis.b   #0FFh, &P3DIR           ; set P3 for output
      mov.w   #STRING, R4                 ; loading the string address into R4
      mov.w   #RESULTS, R8            ; Load RESULTS address into R8
      clr.b   R5                  ; clear R5 (used for storing characters)
      clr.b   R6                  ; clear R6 (used as counter for changes)
      clr.b   R7                  ; clear R7 (length counter) this is extra, dont need the length count

LOOP:
      inc.w   R7                  ; Increment the string length counter
      mov.b   @R4+, R5                ; Load character from string into R5
      cmp     #0x00, R5               ; Check if null character
      jeq     DAEND                   ; If null, jump to end

      cmp.b   #'A', R5                ; Compare with 'A'
      jl      THANKU_NEXT             ; If less than 'A', skip to next character
      cmp.b   #'Z', R5                ; Compare with 'Z'
      jge     THANKU_NEXT             ; If greater than 'Z', skip to next character

      add.b   #32, R5                 ; Convert to lowercase by adding 32
      inc.b   R6                  ; Increment change counter

THANKU_NEXT:
      mov.b   R5, 0(R8)               ; Store the converted character in RESULTS buffer
      inc.w   R8                  ; Increment R8 to point to the next byte in RESULTS
      jmp     LOOP                    ; Continue loop to next character

DAEND:
      mov.b   R6, &P3OUT              ; Output the number of changes to P3
      bis.w   #LPM4, SR               ; Enter low power mode
      nop                         ; Required for debugger
;-------------------------------------------------------------------------------
; STACK POINTER DEFINITION
;-------------------------------------------------------------------------------
      .global __STACK_END
      .sect   .stack
;-------------------------------------------------------------------------------
; INTERRUPT VECTORS
;-------------------------------------------------------------------------------
      .sect   ".reset"                ; MSP430 RESET Vector
      .short  RESET
      .end
```