Cameron Neville

EEEE420 Embedded Systems Design

Lab #3

**Part 1**

Assembly Commands:

bic.w #0xFFFF,r12

bis.w #0x12EF,r12

bic.w #0xFFFF,r13

bic.w #0xFFFF,r14

bic.w #0xFFFF,r15

bis.b r12,r13

bis.w r12,r14

bis.w r12,r15

The commands bic and bis are used to clear or set (respectively) individual bits. They will only change bits that correspond to a 1 in SRC. Calling bic with a SRC of #FFFF is equivalent to calling clr. Calling bis is equivalent to calling move, but only if the previous value of DST is 0x0000 (or if storing 0xFFFF).

**Part 2**

* *symbol* **.equ** *value*
  1. Allows *symbol* to be used in place of *value* in the assembly source
* .**cdecls** [*options* ,]"*filename*"[, "*filename2*"[, ...]
  1. Allows the assembler to pull headers from C code
* **.bss** *symbol*, *size in bytes*[, *alignment*]
  1. Reserves space for variables in the .bss section
* **.sect** “*section name*”
  1. Tells the assembler to begin assembling into the named section
* **.text**
  1. Tells the assembler to begin assembling into the .text section
* **.end**
  1. Can be used to terminate assembly (optional)
* **.byte** *value1*[, ... , *valuen* ]
  1. Places one or more values into consecutive bytes of the current section.
* **.word** *value1*[, ... , *valuen* ]
  1. Initializes one or more 16-bit integers in memory
* **.short** *value1*[, ... , *valuen* ]
  1. Initializes one or more 16-bit integers (halfword) in memory

**Part 3**

The program begins by allocating space in memory for a 16 value array (representing a 4x4 2D array) and a 16 bit “accumulator”. Registers are used to store the current row, column, and index within the array. The program then initializes by moving 0xFF into every position in the array. The program then iterates through the array using two nested loops. First it loops through rows, then loops through each column (or element) within each row. For each element, the program stores the current array index at that spot then adds the index to the accumulator. It then updates the column and index registers to move to the next element. If it has reached the end of the row, it jumps to the “NEXTROW” section of code, which updates to the next row. If it has finished the final row, it jumps to after the loop and restarts.

* .equ (17, 18, 19)
* .cdecls (21)
* .bss (25-28, 30)
* .sect (36, 92, 100)
* .byte (38-41)
* .text (45)
* .retain (46)
* .retainrefs (48)
* .global (91)
* .short (101)

The code is stored from address 0xC000 to 0xC042, which is 43 bytes.

**Part 4**

* Register Direct
  + SRC: 77-78
  + DST: 59-61, 64, 67, 70, 72, 75, 79-80, 83, 84
* Indexed
  + SRC: N/A
  + DST: 66, 77
* Register Indirect
  + SRC: N/A
  + DST: N/A
* Symbolic
  + SRC: N/A
  + DST: 68, 73, 76, 78, 81, 85, 87
* Immediate
  + SRC: 64, 67, 72, 75, 79-80, 84
* Absolute
  + SRC: 66
  + DST: 62

**Part 5**

* i: 0x03D8
* j: 0x03DA
* SUM: 0x03DC
* ARRAY: 0x03DE – 0x03FC

The code is stored from 0xBFEE to 0xC158, which is 363 bytes. It is much larger because the assembly code is fully optimized to the MSP430 and the exact requirements of our program.