Sherman Lam
E155
October 6, 2014

Lab 4 Report: Microcontroller Sorting

1 Introduction

In this lab, I implemented bubble sort in MIPS assembly. This was simulated in MPLAB-X and then tested on the PIC32 microcontroller located on the μMudd32.

2 Design and Testing Methodology

2.1 Sorting

Instead of implementing a max finder for part c of the lab and then use selection sort for part d of the lab, I chose to implement bubble sort for part d. I then simply used the sorting algorithm to find the max for part c of the lab. While this is not the most efficient way (from a coding standpoint) to find the max of a list, it simplified the lab to use one main piece of source code. In addition, bubble sort is simpler to implement than selection sort.

Bubble sort is a very simple sorting algorithm. It is slow for sorting large lists but since we are sorting at most a list of 12 numbers, this algorithm is sufficient. In C, bubble sort is implemented as follows.

```
1  bool flag;                              //1 = sorted, 0 = not sorted
2  int length = 12;                        //length of the list
3  while (flag != 1){                      //keep sorting if not sorted
4      flag = 1;
5      for (int i=0; i<length-1; i++){
6          n1 = array[i];                  //assume the list exists somewhere
7          n2 = array[i+1];
8          if (n1 > n2){
9              pass;
10         }
11         else{                           //swap if they are out of order
12             array[i] = n2;
13             array[i+1] = n1;
14             flag = 0;                   //signal that list isn't sorted
15         }
16     }
17 }
```

2.2 Using LEDs

In order to control the LED with the PIC32, HIGH and LOW logic levels must be written to pins RD1 through RD7. These 8 pins are connected to the 8 lower LEDs on the μMudd32. On the PIC32, pins RD1 through RD7 are part of a larger group of pins. Microchip gave this group of pins the designation "PORTD". In memory, the state of PORTD is simply stored as a 32-bit number. To change the outputs of any pin in PORTD, the PORTD bits

in memory can be written to either 1 or 0. Notice though that while PORTD is stored as a 32-bit number, Figure 1 shows that there are only 16 pins in PORTD and so only the bottom 16 bits of PORTD are used.

However, to use these pins they must first be set to outputs or inputs. Those who have used an Arduino before will be familiar with this concept. On an Arduino, all pins must be set to inputs or outputs prior to use with the "pinmode" command.

PIC32 has a similar method of doing this. However, it operates at a lower level than Arduino. This is TRIS. Specifically, TRISD is used to control the mode of the pins in PORTD. TRISD is also a 32-bit number but only the bottom 16 bits are used (see Figure 1 for the pin-to-bit mapping). When a bit in TRISD is 0, the corresponding pin is configured as an output, and vice versa (see PIC32 datasheet).

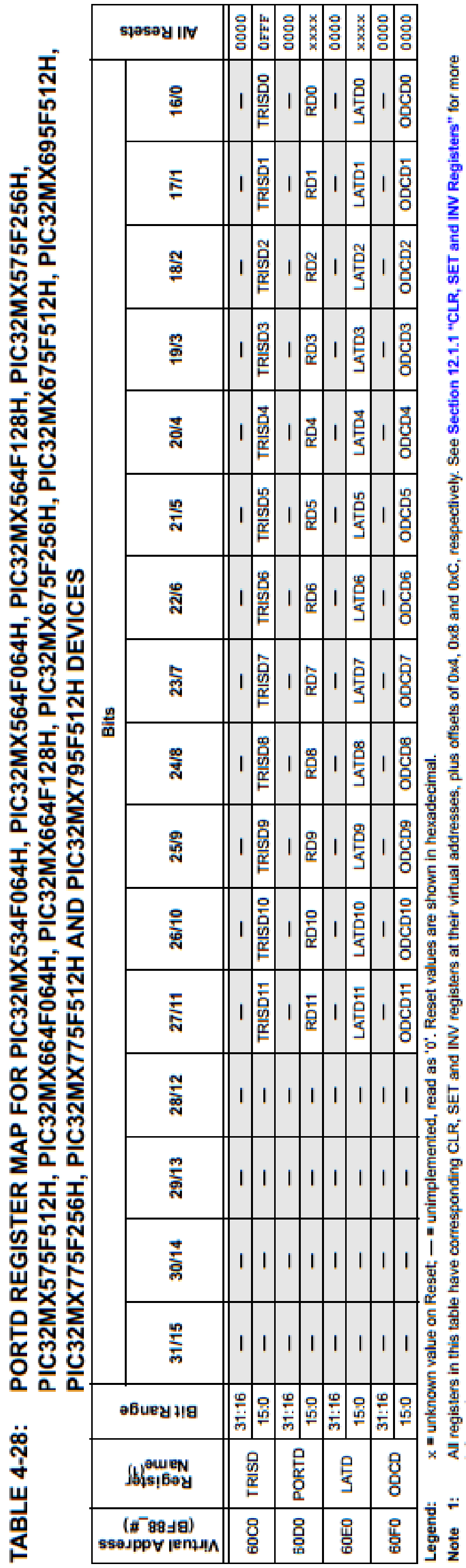

TABLE 4-28: PORTD REGISTER MAP FOR PIC32MX534F064H, PIC32MX564F064H, PIC32MX564F128H, PIC32MX575F256H, PIC32MX575F512H, PIC32MX664F064H, PIC32MX664F128H, PIC32MX675F256H, PIC32MX675F512H, PIC32MX695F512H, PIC32MX775F256H, PIC32MX775F512H AND PIC32MX795F512H DEVICES

Virtual Address (BF88_#)	Register Name[(1)]	Bit Range	Bits																All Resets
			31/15	30/14	29/13	28/12	27/11	26/10	25/9	24/8	23/7	22/6	21/5	20/4	19/3	18/2	17/1	16/0	
60C0	TRISD	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	—	—	—	—	TRISD11	TRISD10	TRISD9	TRISD8	TRISD7	TRISD6	TRISD5	TRISD4	TRISD3	TRISD2	TRISD1	TRISD0	0FFF
60D0	PORTD	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	—	—	—	—	RD11	RD10	RD9	RD8	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	xxxx
60E0	LATD	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	—	—	—	—	LATD11	LATD10	LATD9	LATD8	LATD7	LATD6	LATD5	LATD4	LATD3	LATD2	LATD1	LATD0	xxxx
60F0	ODCD	31:16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0000
		15:0	—	—	—	—	ODCD11	ODCD10	ODCD9	ODCD8	ODCD7	ODCD6	ODCD5	ODCD4	ODCD3	ODCD2	ODCD1	ODCD0	0000

Legend: x = unknown value on Reset; — = unimplemented, read as '0'. Reset values are shown in hexadecimal.

Note 1: All registers in this table have corresponding CLR, SET and INV registers at their virtual addresses, plus offsets of 0x4, 0x8 and 0xC, respectively. See **Section 12.1.1 "CLR, SET and INV Registers"** for more information.

Figure 1: Table from datasheet which shows which values of TRISD and PORTD are used. This also shows which pins PORTD is mapped to.

2.2.1 Simulation

The following are some of the tests I ran to verify that the program was running correctly. On the left of each image is the list in the stack before it was sorted. The list on the right is the sorted list. Note that due to MPLAB and MPLAB-X's memory viewing system, larger addresses are towards the bottom. Since the stack grow from large to small addresses, the "top" of the stack is at the bottom of the image. So sorted lists will have the smallest number at the highest address / the bottom of the image / the top of the stack.

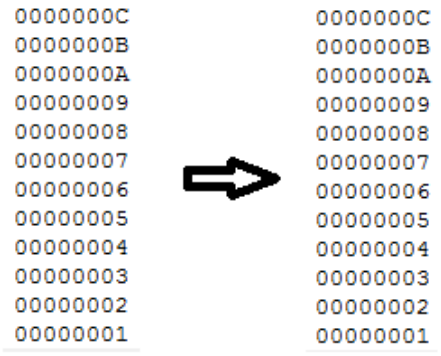

Figure 2: A sorted list should remain sorted.

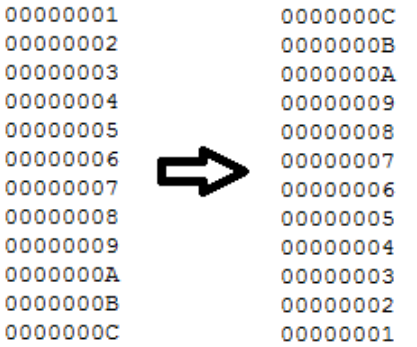

Figure 3: The sort works on a inverted list.

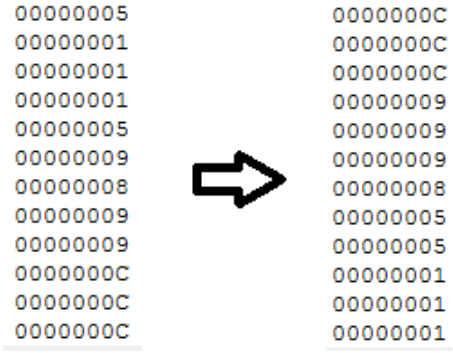

Figure 4: The sort works on a list of randomly selected numbers with repeats.

16373	A000_FFD0	FFFFFFFF
16374	A000_FFD4	00000002
16375	A000_FFD8	00000003
16376	A000_FFDC	00000004
16377	A000_FFE0	00000005
16378	A000_FFE4	00000006
16379	A000_FFE8	00000007
16380	A000_FFEC	00000008
16381	A000_FFF0	00000009
16382		0000000A
16383		0000000B
16384		0000000C

16373	A000_FFD0	0000000C
16374	A000_FFD4	0000000B
16375	A000_FFD8	0000000A
16376	A000_FFDC	00000009
16377	A000_FFE0	00000008
16378	A000_FFE4	00000007
16379	A000_FFE8	00000006
16380	A000_FFEC	00000005
16381	A000_FFF0	00000004
16382		00000003
16383		00000002
16384		FFFFFFFF

Figure 5: The sorts works on lists with negative numbers (in two's compliment).

3 Technical Documentation

3.1 MIPS Code - Finding the Max

```
1  /*
2  This uses bubble sort to sort a list of 5, 32-bit numbers. It then
3  writes the largest number to the LEDs.
4
5  Author: Sherman Lam
6  Date: 10/5/14
7  Email: slam@g.hmc.edu
8  */
9
10 #include <P32xxxx.h>    # header file that defines TRISD and PORTD
11
12 .global main           # start the main function
13
14 # Compiler instructions
15 .text           # store the code in the main program section of RAM
16 .set noreorder  # do not let the compiler reorganize your code
17
18 # define globals
19 #define LST1a   0x0
20 #define LST1b   0x1
21 #define LST2a   0x0
22 #define LST2b   0x2
23 #define LST3a   0x0
24 #define LST3b   0x3
25 #define LST4a   0x0
26 #define LST4b   0x4
27 #define LST5a   0x0000
28 #define LST5b   0x000F
29
30 #define PINMODE     0x0F00       # 1 = input, 0 = output
31
32
33 # Register use:
34 #   $t0 = counter (i)
35 #   $t1 = i*4
36 #   $t2 = slt results
37 #   $t3 = lower number (n1)
38 #   $t4 = upper number (n2)
39 #   $t5 = flag. 1 if list is sorted. 0 if not.
40 #   $t6 = 4
41 #   $t7 = stack pointer + i*4
42 #   $t8 = address of PORTD and TRISD
43 #   $s0 =
44 #   $s1 = number to load / store to memory
45
46 .ent main
47
48 main:
49     # load 5 numbers into the stack
50     lui     s1, LST1a       # load first half of the number into reg
51     ori     s1, LST1b       # load second half of the number into reg
52     addi    sp, sp, -4      # add mem
53     sw      s1, 0(sp)       # load number into ram
54     lui     s1, LST2a       # load first half of the number into reg
```

```
55      ori     s1, LST2b       # load second half of the number into reg
56      addi    sp, sp, -4      # add mem
57      sw      s1, 0(sp)       # load number into ram
58      lui     s1, LST3a       # load first half of the number into reg
59      ori     s1, LST3b       # load second half of the number into reg
60      addi    sp, sp, -4      # add mem
61      sw      s1, 0(sp)       # load number into ram
62      lui     s1, LST4a       # load first half of the number into reg
63      ori     s1, LST4b       # load second half of the number into reg
64      addi    sp, sp, -4      # add mem
65      sw      s1, 0(sp)       # load number into ram
66      lui     s1, LST5a       # load first half of the number into reg
67      ori     s1, LST5b       # load second half of the number into reg
68      addi    sp, sp, -4      # add mem
69      sw      s1, 0(sp)       # load number into ram
70
71      # pin configuration
72      la      t8, TRISD           # load the address of TRISD into t8
73      addi    s1, zero, PINMODE   # load the pinmodes into s1
74      sw      s1, 0(t8)           # set which pins are inputs or outputs.
75
76      add     t5, zero, 0     # start a flag that indicates if the list is sorted.
77      addi    t6, zero, 4     # load 4 into $t6
78  while:                      # start of while loop
79      bne     t5, zero, done  # if $t5!=0, the list is sorted
80      nop
81      addi    t5, zero, 1     # set the flag to sorted
82      add     t0, zero, zero  # init i at 0
83  for:                        # start of for loop
84      sltiu   t2, t0, 4       # make sure that the counter has not exceeded the length of the lis
85      beq     t2, zero, while # if the for loop is over, jump back to the while loop
86      nop
87      mul     t1, t0, t6      # calc i*4
88      add     t7, sp, t1      # calc stack pointer + i*4
89      lw      t3, 0(t7)       # load the number stored at the top of the stack (lowest addr) [n1]
90      lw      t4, 4(t7)       # load the 2nd to last number [n2]
91      add     t0, t0, 1       # increment the counter
92      slt     t2, t3, t4      # check if n1 < n2 (out of order)
93      beq     t2, zero, for   # pass if the numbers are in order ($t2=0)
94      nop
95      sw      t3, 4(t7)       # put n1 in n2's spot
96      sw      t4, 0(t7)       # put n2 in n1's spot
97      add     t5, zero, zero  # set the flag to 0 since the list is not sorted
98      j       for
99      nop
100 done:
101     lw      s1, 0(sp)       # load the largest number in $s1
102     la      t8, PORTD       # load the address of PORTD into $t8
103     sw      s1, 0(t8)       # store the number in PORTD -> write to LEDs
104     jr      ra              # return to function call
105     nop
106
107 .end main
```

3.2 MIPS Code - Sorting a List

```
1  /*
2  This uses bubble sort to sort a list of 12, 32-bit numbers
3
```

```
4  Author: Sherman Lam
5  Date: 10/5/14
6  Email: slam@g.hmc.edu
7  */
8
9  .global main        # start the main function
10
11 # Compiler instructions
12 .text              # store the code in the main program section of RAM
13 .set noreorder     # do not let the compiler reorganize your code
14
15 # define globals
16 #define LST1a   0x0     # top 16 bits of the number
17 #define LST1b   0xC     # bottom 16 ibts of the number
18 #define LST2a   0x1     # top 16 bits of the number
19 #define LST2b   0xB     # bottom 16 ibts of the number
20 #define LST3a   0x0     # top 16 bits of the number
21 #define LST3b   0xA     # bottom 16 ibts of the number
22 #define LST4a   0x0     # top 16 bits of the number
23 #define LST4b   0x9     # bottom 16 ibts of the number
24 #define LST5a   0x0     # top 16 bits of the number
25 #define LST5b   0x8     # bottom 16 ibts of the number
26 #define LST6a   0x0     # top 16 bits of the number
27 #define LST6b   0x7     # bottom 16 ibts of the number
28 #define LST7a   0x0     # top 16 bits of the number
29 #define LST7b   0x6     # bottom 16 ibts of the number
30 #define LST8a   0x0     # top 16 bits of the number
31 #define LST8b   0x5     # bottom 16 ibts of the number
32 #define LST9a   0x0     # top 16 bits of the number
33 #define LST9b   0x4     # bottom 16 ibts of the number
34 #define LST10a  0x0     # top 16 bits of the number
35 #define LST10b  0x3     # bottom 16 ibts of the number
36 #define LST11a  0x0     # top 16 bits of the number
37 #define LST11b  0x2     # bottom 16 ibts of the number
38 #define LST12a  0xFFFF    # top 16 bits of the number
39 #define LST12b  0xFFFF    # bottom 16 ibts of the number
40
41 # Register use:
42 #   $t0 = counter (i)
43 #   $t1 = i*4
44 #   $t2 = slt results
45 #   $t3 = lower number (n1)
46 #   $t4 = upper number (n2)
47 #   $t5 = flag. 1 if list is sorted. 0 if not.
48 #   $t6 = 4
49 #   $t7 = ADDR + i*4
50 #   $t8 =
51 #   $s0 =
52 #   $s1 = number
53
54 .ent main
55
56 main:
57     # load 12 numbers into the stack
58     lui     $s1, LST1a      # load first half of the number into reg
59     ori     $s1, LST1b      # load second half of the number into reg
60     addi    $sp, $sp, -4    # add mem
61     sw      $s1, 0($sp)     # load number into ram
62     lui     $s1, LST2a      # load first half of the number into reg
```

```
63      ori     $s1, LST2b      # load second half of the number into reg
64      addi    $sp, $sp, -4    # add mem
65      sw      $s1, 0($sp)     # load number into ram
66      lui     $s1, LST3a      # load first half of the number into reg
67      ori     $s1, LST3b      # load second half of the number into reg
68      addi    $sp, $sp, -4    # add mem
69      sw      $s1, 0($sp)     # load number into ram
70      lui     $s1, LST4a      # load first half of the number into reg
71      ori     $s1, LST4b      # load second half of the number into reg
72      addi    $sp, $sp, -4    # add mem
73      sw      $s1, 0($sp)     # load number into ram
74      lui     $s1, LST5a      # load first half of the number into reg
75      ori     $s1, LST5b      # load second half of the number into reg
76      addi    $sp, $sp, -4    # add mem
77      sw      $s1, 0($sp)     # load number into ram
78      lui     $s1, LST6a      # load first half of the number into reg
79      ori     $s1, LST6b      # load second half of the number into reg
80      addi    $sp, $sp, -4    # add mem
81      sw      $s1, 0($sp)     # load number into ram
82      lui     $s1, LST7a      # load first half of the number into reg
83      ori     $s1, LST7b      # load second half of the number into reg
84      addi    $sp, $sp, -4    # add mem
85      sw      $s1, 0($sp)     # load number into ram
86      lui     $s1, LST8a      # load first half of the number into reg
87      ori     $s1, LST8b      # load second half of the number into reg
88      addi    $sp, $sp, -4    # add mem
89      sw      $s1, 0($sp)     # load number into ram
90      lui     $s1, LST9a      # load first half of the number into reg
91      ori     $s1, LST9b      # load second half of the number into reg
92      addi    $sp, $sp, -4    # add mem
93      sw      $s1, 0($sp)     # load number into ram
94      lui     $s1, LST10a     # load first half of the number into reg
95      ori     $s1, LST10b     # load second half of the number into reg
96      addi    $sp, $sp, -4    # add mem
97      sw      $s1, 0($sp)     # load number into ram
98      lui     $s1, LST11a     # load first half of the number into reg
99      ori     $s1, LST11b     # load second half of the number into reg
100     addi    $sp, $sp, -4    # add mem
101     sw      $s1, 0($sp)     # load number into ram
102     lui     $s1, LST12a     # load first half of the number into reg
103     ori     $s1, LST12b     # load second half of the number into reg
104     addi    $sp, $sp, -4    # add mem
105     sw      $s1, 0($sp)     # load number into ram
106
107     add     $t5, $0, $0     # start a flag that indicates if the list is sorted.
108     addi    $t6, $0, 4      # load 4 into $t6
109 while:                      # start of while loop
110     bne     $t5, $0, done   # if $t5!=0, the list is sorted
111     nop
112     addi    $t5, $0, 1      # set the flag to sorted
113     add     $t0, $0, $0     # init i at 0
114 for:                        # start of for loop
115     sltiu   $t2, $t0, 11    # make sure that the counter has not exceeded the length of the lis
116     beq     $t2, $0, while  # if the for loop is over, jump back to the while loop
117     nop
118     mul     $t1, $t0, $t6   # calc i*4
119     add     $t7, $sp, $t1   # calc stack pointer + i*4
120     lw      $t3, 0($t7)     # load the number stored at the top of the stack (lowest addr) [n1]
121     lw      $t4, 4($t7)     # load the 2nd to last number [n2]
```

```
122     add     $t0, $t0, 1     # increment the counter
123     slt     $t2, $t3, $t4   # check if n1 < n2 (out of order)
124     beq     $t2, $0, for    # pass if the numbers are in order ($t2=0)
125     nop
126     sw      $t3, 4($t7)     # put n1 in n2's spot
127     sw      $t4, 0($t7)     # put n2 in n1's spot
128     add     $t5, $0, $0     # set the flag to 0 since the list is not sorted
129     j       for
130     nop
131 done:
132
133     jr      $ra             # return to function call
134     nop
135
136 .end main
```

4 Results and Discussion

The sorting works! For part c of the lab, the max number is successfully displayed on the LEDs. For part d of the lab, the sorted list can be monitored through MPLAB-X's memory viewer.

5 Conclusion

5.1 Time Spent

Programming, Simulating 5hrs

Writing Report 2.5hrs

Total Time Spent 7.5hrs

5.2 Suggestions for lab

Use MPLAB-X instead of MPLAB. The interface is easier to use, cleaner, and not as buggy. Writing a lab to use MPLAB-X would be helpful.