Mitchell Scott
September 2, 2021
Lab 2
ASEN 5067

1) The options we added to the project will: (Wl) pass a list of args to the linker, (-p) specify the addresses for Psects. For example passing -presetVec=0h will put the instruction for reset at the address 0x0.

2) The RADIX directive tells the machine what base to interpret constants in. e.g. we specify Dec in our asm file so any constants will be read as base 10. The config option for FOSC determines the source of the oscillator. The original setting was HS1, meaning high speed crystal resonator at medium speed (4-16MHz). The other option INTIO1 tells the machine to use the internal oscillator which will use pin RA6 as output (meaning we shouldn't use it).

3) The mainline program is stored at address 0x1c. The initial subroutine is at 0x22.

4) Binary: 0110 1110 1001 0101

   Hex: 0x6E

   a: 0b

   f: 0x95

5) Binary: 0110 1010 1000 1100

   Hex: 0x6A

   a: 0b

   f: 0x8C

6) LATD: 0xF8C

   TRISD: 0xF95

7) Instruction at 0x22 moves 0xBD into WREG, 0x24 moves the value of WREG into TRISD (portD's reader), 0x26 then puts LATD (portD's writer/config) to 0x00

8) 0x1E calls BTG a bit toggling instruction. This call sets the third bit (from the right) of

LATD, leaving it with the value 0x04.

9)

Before	After
TRISD: 0xBD	TRISD: 0xFF
WREG: 0xBD	WREG: 0xBD
LATD: 0x04	LATD: 0x00

TRISD and LATD both changed because the program was reset. When resetting, the program includes a file xc.inc which most likely contains defines from the manufacturer about what everything should be (they define the TRISD mnemonic).

10)

The status register went from 0x00 to 0x04 when the CLRF happened. This is because the STATUS register shows us the status of an ALU operation. LATD gets set to zero so the zero bit is on.

N - (bit 4) Negative result

OV - (bit 3) overflow

Z - (bit 2) Zero bit

DC - (bit 1) digit carry bit

C - (bit 0) carry bit

11)

The STATUS register stays the same when we execute a MOVLW. The move puts a literal value into WREG. The move is not an arithmetic operation so the STATUS register is unchanged.

12)

The STATUS register is set to zero after INCF. This is because we are using the ALU to add a value to value in WREG and storing it back in WREG. The increment operation does not set any of the STATUS bits because it is not supposed to. If you press f7 as many times as the size of the register value (8-bit = 256 times) you will see an overflow

bit set.

13)

After pressing f7 a bunch of times I saw the DC bit set before the OV. The overflow bit only changes in signed arithmetic when the MSB (sign bit) is affected by the operation. The DC bit is a carry over the lower 4 bits (00001111 -> 00010000).

14)

The second bit of the STATUS register is the digit carry bit. The DC bit is a carry over the lower 4 bits (00001111 -> 00010000). (ie there was an overflow from the lower nibble to the upper nibble).

15)

If you press f5 enough the STATUS will become 0x1A meaning that WREG has its MSB set, experienced an overflow and a digit carry. That is what happens when WREG goes from 0x7F to 0x80.

16)

Incrementing WREG from 0x80 to 0x81 technically results in a negative number so the STATUS register is set to 0x10 (the negative bit is set).

17)

When WREG goes from 0xFF to 0x00 STATUS is set to 0x07. This means that WREG is zero, experienced a digit carry and a carry (1111 1111 -> 0000 0000).

18)

The DAW instruction does not affect the WREG when it is 0x02. This is because the DAW instruction handles overflows in BCD formatted values.

19)

WREG starts at 0x01, it is incremented by 0x01 resulting in 0x02. WREG is then added to itself, resulting in 0x04. Then DAW is called and nothing changes, Next we INCF again resulting in WREG = 0x05, then we add WREG to itself resulting in 0x0A. The next step is a DAW call which formats WREG as a BCD (each nibble represents its own digit i.e. 0x0A = 1010 = 10, in BCD 10 is two digits 0x1 and 0x0, like hex without the letters) this results in WREG being 0x10.

20)

The NEGF command uses twos compliment to negate the value in WREG.

0000 0100 -> 1111 1011+1 -> 1111 1100 : 0x04 -> 0xFC

21)

The RLNCF instruction is basically a binary left shift. Something special is that it wraps bits (i.e. 10101010 -> 01010101, most shifts just fill 0s or 1s).

22)

The RLCF command is almost the same as the RLNCF. The major difference is that RLNCF wraps its bits in the result, the RLCF will put carry bits in the carry bit of the STATUS reg.

23)

The DS directive will reserve but not initialize the number of bytes passed to it. I then used the MOVWF instruction because that does exactly what we want, put the WREG value into count.

24)

The address of the variable count is 0x2.

25)

0x6CE8

26)

0x748C

Computational Questions

1)

a) -24
   24 / 2 = 12 R 0
   12 / 2 = 6 R 0
   6 / 2 = 3 R 0
   3 / 2 = 1 R 1
   1 / 2 = 0 R 1
   0001 1000 -> two's compliment -> 1110 0111 -> **1110 1000**

b) -85
85 / 2 = 42 R 1
42 / 2 = 21 R 0
21 / 2 = 10 R 1
10 / 2 = 5 R 0
5 / 2 = 2 R 1
2 / 2 = 1 R 0
1 / 2 = 0 R 1
0101 0101 -> twos comp -> 1010 1010 -> **1010 1011**

2)

a)

32 / 2 = 16 R 0
16 /2 = 8 R 0
8 / 2 = 4 R 0
4 / 2 = 2 R 0
2 / 2 = 1 R 0
1 / 2 = 0 R 1
0010 0000

12 / 2 = 6 R 0
6 / 2 = 3 R 0
3 / 2 = 1 R 1
1 / 2 = 0 R 1
0000 1100 -> tc -> 1111 0011 -> 1111 0100

```
 0010 0000
+1111 0100
  0001 0100  Carry: 1
```

0001 0100 = 16 + 4 = 20

b)

16 / 2 = 8 R 0
8 / 2 = 4 R 0
4 / 2 = 2 R 0
2 /2 = 1 R 0
1 / 2 = 0 R 1
0001 0000

73 / 2 = 36 R 1
36 /2 = 18 R 0
18 / 2 = 9 R0
9 / 2 = 4 R 1
4 / 2 = 2 R 0
2 / 2 = 1 R 0

1 / 2 = 0 R 1
0100 1001 -> tc -> 1011 0110 -> 1011 0111

 0001 0000
+1011 0111
 1100 0111 -> tc -> 0011 1000 -> 0011 1001 -> -57

3)

2.11) The program counter is a register that saves the address of the next instruction to execute. This is why instructions need a fetch and execute (2 cycles).

3.5) The instruction MOVWF 0x7F,1, this will move WREG into the register at 0x7F, however because of the 1 at the end it will use the BSR to select the address bank 0x4.

3.9) 0x89 + 0x77 -> 137 + 119 -> 256. WREG will have 0x00 because of overflow.

4.12)

a) 0xF1
b) A negative and DC (i.e. 0x12)
c) This is the 5th from right and the second from right bits (in STATUS)

4.13)
The sum is 0x39 and the STATUS flags are 0x9 (overflow and carry)

4.1se)

a) 0x8A
b) STATUS -> 0x18 (Neg Overfow)
c) 0x00 (because an overflow was detected)

4.2se)

d) 0x5F
e) STATUS -> 0x00
f) 0x5F (because an overflow wasn't detected)

5.16)

```
MOVLW 0x78
ADDLW 0xF2
```

The result is 0x6A and the carry bit will be set. The carry bit is used for ADDWF, ADDLW, SUBLW and SUBWF instructions. The overflow bit is used for signed arithmetic (when the result is too large for the register) so it will not be set.

5067 only

1)

a)

-346

346 / 2 = 173 R 0

173 / 2 = 86 R 1

86 / 2 = 43 R 0

43 / 2 = 21 R 1

21 / 2 = 10 R 1

10 / 2 = 5 R 0

5 / 2 = 2 R 1

2 / 2 = 1 R 0

1 / 2 = 0 R 1

0001 0101 1010 -> 1110 1010 0101 -> **1110 1010 0110**

b)

-130

130 / 2 = 65 R 0

65 / 2 = 32 R 1

32 / 2 = 16 R 0

16 / 2 = 8 r 0

8 / 2 = 4 R 0

4 / 2 = 2 R 0

2 / 2 = 1 R 0

1 / 2 = 0 R 1

1000 0010 -> 0111 1101 -> **0111 1110**

2)

a)

-55

55 / 2 = 27 R 1

27 / 2 = 13 R 1

13 / 2 = 6 R 1

6 / 2 = 3 R 0

3 / 2 = 1 R 1

1 / 2 = 0 R 1

0011 0111 -> 1100 1000 -> 1100 1001

-16
16 /2 = 8 R 0
8 / 2 = 4 R 0
4 / 2 = 2 R 0
2 / 2 = 1 R 0
1 / 2 = 0 R 1
0001 0000 -> 1110 1111 -> 1111 0000
1100 1001 + 1111 0000 = 1011 1001

b)
-54
54 / 2 = 27 R 0
27 / 2 = 13 R 1
13 / 2 = 6 R 1
6 / 2 = 3 R 0
3 / 2 = 1 R 1
1 / 2 = 0 R 1
0011 0110 -> 1100 1001 -> 1100 1010

28
28 / 2 = 14 R 0
14 /2 = 7 R0
7/ 2 = 3 R 1
3 / 2 = 1 R 1
1 / 2 = 0 R 1
0001 1100

1100 1010 + 0001 1100 = 1110 0110 -> -26

3.19) Most instructions are simple operations like add, sub, mov; these things can all happen in a single instruction cycle. This means that in one cycle the computer will be given an instruction, say mov, a location and data. Then it only takes a single cycle for the operation to actually happen.

4.4se)

g) 0x7F
h) STATUS -> 0x09 (overflow and carry)
i) 0x00 (overflow was detected)

4.5se)

Hex sum: 0x7559

5.17)

The carry bit is used in ADDWF, ADDLW, SUBLW and SUBWF instructions. The overflow bit is used for signed values, if the sum of two numbers is greater than the size of

the register the overflow bit gets set (like when adding 0xFF to 0xFF).