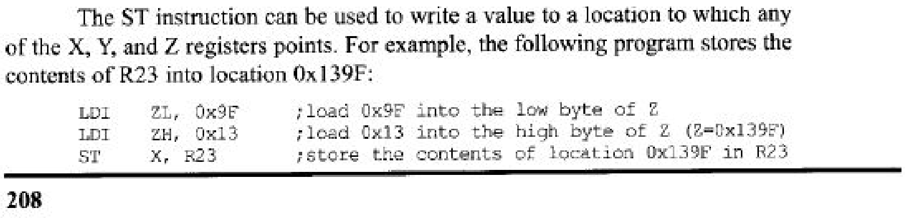
Gmail by Google

Gary Hill <hellogaryhill@gmail.com>

RE: EE346 Questions about Indirect Addressing

4 messages

Gilbert Tse <tse.gilbert@gmail.com> Wed, Nov 6, 2013 at 6:52 PM
To: Gary Hill <hellogaryhill@gmail.com>

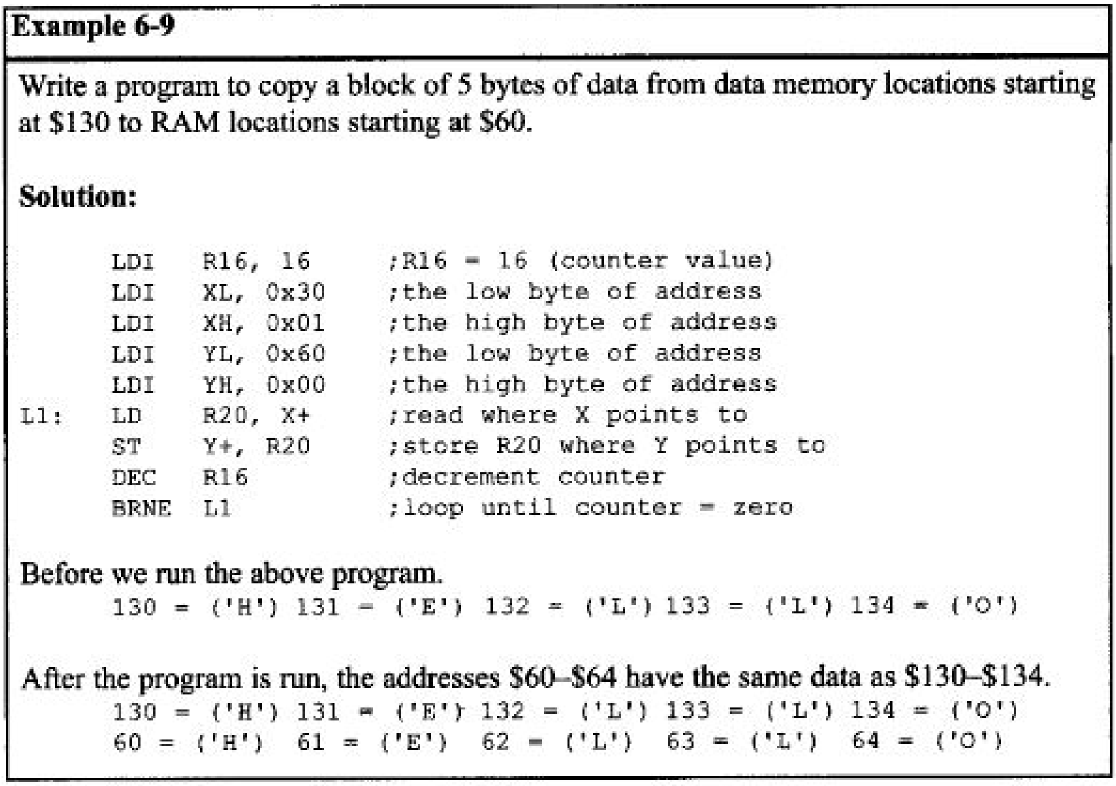


The ST instruction can be used to write a value to a location to which any of the X, Y, and Z registers points. For example, the following program stores the contents of R23 into location 0x139F:

```
LDI    ZL, 0x9F       ;load 0x9F into the low byte of Z
LDI    ZH, 0x13       ;load 0x13 into the high byte of Z (Z=0x139F)
ST     X, R23         ;store the contents of location 0x139F in R23
```

208

In the above question I am wondering why we are storing to X? Shouldn't it be Z if that is the location we are loading into Z?



Example 6-9

Write a program to copy a block of 5 bytes of data from data memory locations starting at $130 to RAM locations starting at $60.

Solution:

```
      LDI   R16, 16     ;R16 = 16 (counter value)
      LDI   XL, 0x30    ;the low byte of address
      LDI   XH, 0x01    ;the high byte of address
      LDI   YL, 0x60    ;the low byte of address
      LDI   YH, 0x00    ;the high byte of address
L1:   LD    R20, X+     ;read where X points to
      ST    Y+, R20     ;store R20 where Y points to
      DEC   R16         ;decrement counter
      BRNE  L1          ;loop until counter = zero
```

Before we run the above program.

```
      130 = ('H') 131 = ('E') 132 = ('L') 133 = ('L') 134 = ('O')
```

After the program is run, the addresses $60–$64 have the same data as $130–$134.

```
      130 = ('H') 131 = ('E') 132 = ('L') 133 = ('L') 134 = ('O')
      60 = ('H')  61 = ('E')  62 = ('L')  63 = ('L')  64 = ('O')
```

In this example, I am confused as to why we set 16 as the counter value. I am pretty dumb when it comes to conversions, but how is it that we know the end address will be at 134? 5 bytes is 2.5 words...we can't have half a word? so 3 words? then 130+3 = 133? I'm not sure.

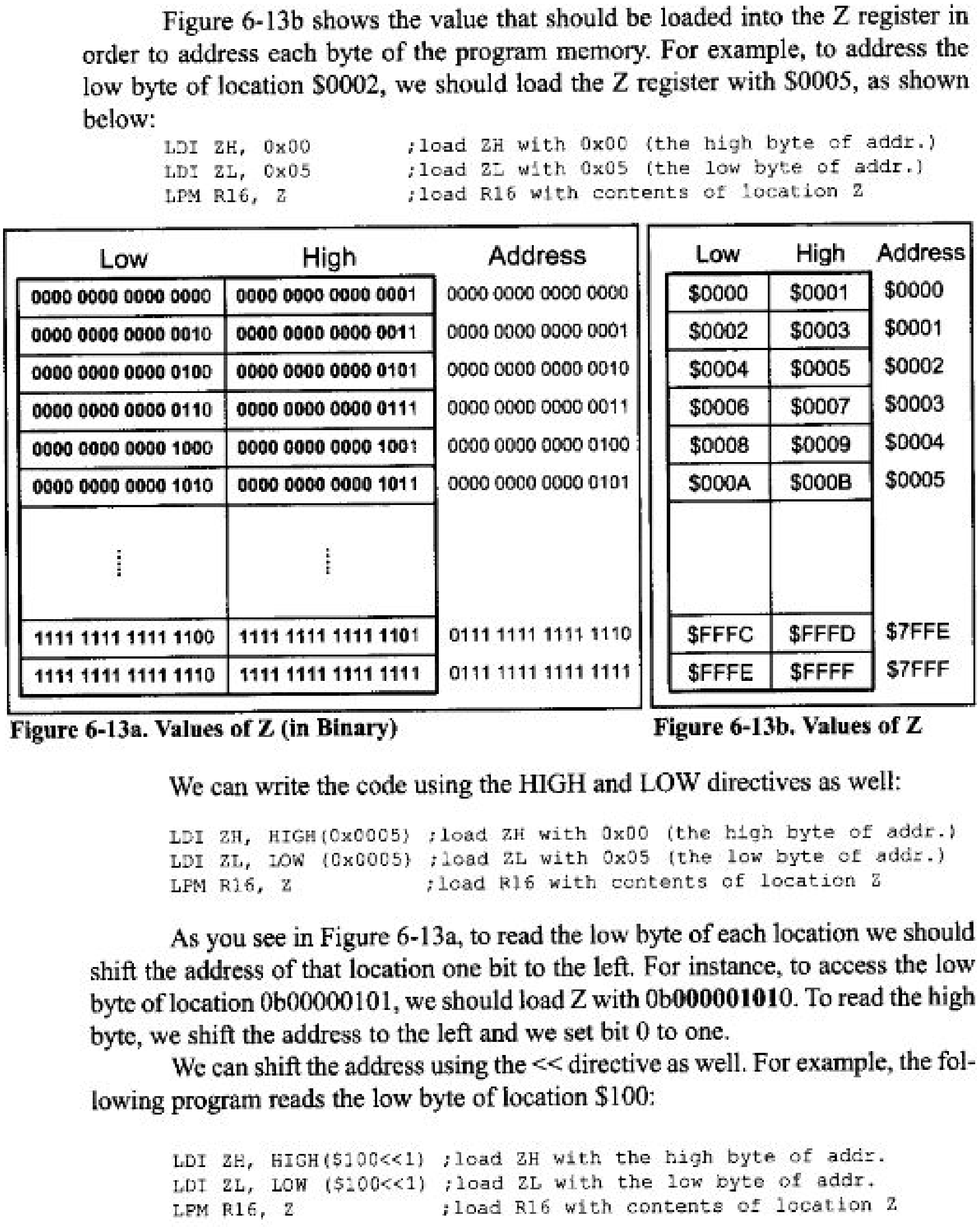


Figure 6-13b shows the value that should be loaded into the Z register in order to address each byte of the program memory. For example, to address the low byte of location $0002, we should load the Z register with $0005, as shown below:

```
LDI ZH, 0x00        ;load ZH with 0x00 (the high byte of addr.)
LDI ZL, 0x05        ;load ZL with 0x05 (the low byte of addr.)
LPM R16, Z          ;load R16 with contents of location Z
```

Low	High	Address
0000 0000 0000 0000	0000 0000 0000 0001	0000 0000 0000 0000
0000 0000 0000 0010	0000 0000 0000 0011	0000 0000 0000 0001
0000 0000 0000 0100	0000 0000 0000 0101	0000 0000 0000 0010
0000 0000 0000 0110	0000 0000 0000 0111	0000 0000 0000 0011
0000 0000 0000 1000	0000 0000 0000 1001	0000 0000 0000 0100
0000 0000 0000 1010	0000 0000 0000 1011	0000 0000 0000 0101
⋮	⋮	
1111 1111 1111 1100	1111 1111 1111 1101	0111 1111 1111 1110
1111 1111 1111 1110	1111 1111 1111 1111	0111 1111 1111 1111

Figure 6-13a. Values of Z (in Binary)

Low	High	Address
$0000	$0001	$0000
$0002	$0003	$0001
$0004	$0005	$0002
$0006	$0007	$0003
$0008	$0009	$0004
$000A	$000B	$0005
$FFFC	$FFFD	$7FFE
$FFFE	$FFFF	$7FFF

Figure 6-13b. Values of Z

We can write the code using the HIGH and LOW directives as well:

```
LDI ZH, HIGH(0x0005) ;load ZH with 0x00 (the high byte of addr.)
LDI ZL, LOW (0x0005) ;load ZL with 0x05 (the low byte of addr.)
LPM R16, Z           ;load R16 with contents of location Z
```

As you see in Figure 6-13a, to read the low byte of each location we should shift the address of that location one bit to the left. For instance, to access the low byte of location 0b00000101, we should load Z with 0b**000001010**. To read the high byte, we shift the address to the left and we set bit 0 to one.

We can shift the address using the << directive as well. For example, the following program reads the low byte of location $100:

```
LDI ZH, HIGH($100<<1) ;load ZH with the high byte of addr.
LDI ZL, LOW ($100<<1) ;load ZL with the low byte of addr.
LPM R16, Z            ;load R16 with contents of location Z
```

I'm also a little confused with this. I follow the directions under the Figure, because it makes sense to shift one bit to left if we want the LOW byte. but above it says that they want to load the LOW byte of $0002. According to the table the low byte is 0004, not 0005. So why did they load 0005?

Gary Hill <hellogaryhill@gmail.com> Thu, Nov 7, 2013 at 10:05 AM
To: Gilbert Tse <tse.gilbert@gmail.com>

The LD is working with SRAM whose word size is 8 bits (not 16 bits). Therefore 5 bytes.
[Quoted text hidden]

Gilbert Tse <tse.gilbert@gmail.com> Thu, Nov 7, 2013 at 12:44 PM
To: Gary Hill <hellogaryhill@gmail.com>

Got it. What about for the first and last problems?

Thanks in advance.

[Quoted text hidden]

Gary Hill <hellogaryhill@gmail.com> Thu, Nov 7, 2013 at 12:52 PM
To: Gilbert Tse <tse.gilbert@gmail.com>

You are correct in both cases. As mentioned in class the authors have adopted a big endian convention and are trying to apply it to a little endian architecture, which seems to confuse even them. This is unfortunate from the student's perspective. You may want to remind me about this in class so I can explain what is happening

[Quoted text hidden]