Jose Franco Baquera

February 21, 2019

Assignment 2

CS 473: Architectural Concepts I

1.

a. For the following C statement, what is the corresponding MIPS assembly code? Consider g, f as $t1, $t0 respectively (i.e. g = $t1 and f = $t0).
f = -g – f;

NOTE: $zero represents the $0 register.

The corresponding MIPS assembly code is the following (one version of many possible ways):

```
sub $t2, $zero, $t1 # Store -g in temporary register $t2.
sub $t0, $t2, $t0 # Subtract f from -g and store it in variable f.
```

It is important to note that the first line "negates" g.

b. For the following C statement, what is the corresponding MIPS assembly code? Consider g, f as $t1, $t0 respectively (i.e. g = $t1 and f = $t0).
f = g + (-f – 5);

NOTE: $zero represents the $0 register.

The corresponding MIPS assembly code is the following (one version of many possible ways):

```
sub $t2, $zero, $t0  # Store -f in temporary register $t2.
addi $t2, $t2, -5 # Storing -f-5 in temporary register $t2.
add $t0, $t1, $t2 # Storing g+(-f-5) in variable f.
```

It is important to note that the first line "negates" f.

2.

a. Translate 0xabcdef12 into decimal.

NOTE: Assume we are only interested in the equivalent unsigned decimal number.

$$
\begin{aligned}
0xabcdef12 &= 10*16^7 + 11*16^6 + 12*16^5 + 13*16^4 + 14*16^3 + 15*16^2 + 1*16^1 + 2*16^0 \\
&= 2684354560 + 184549376 + 12582912 + 851968 + 57344 + 3840 + 16 + 2 \\
&= 2882400018
\end{aligned}
$$

ANSWER: 2882400018

b. Translate 0x10203040 into decimal.

NOTE: Assume we are only interested in the equivalent unsigned decimal number.

$$
\begin{aligned}
0x10203040 &= 1*16^7 + 0*16^6 + 2*16^5 + 0*16^4 + 3*16^3 + 0*16^2 + 4*16^1 + 0*16^0 \\
&= 268435456 + 0 + 2097152 + 0 + 12288 + 0 + 64 + 0 \\
&= 270544960
\end{aligned}
$$

ANSWER: 270544960

3. For the MIPS assembly instructions below, what is the corresponding C statement?

```
max:
lw      $t0, 0($a0)       #load the first array value into $t0
addi    $t1, $0, 1        #intialize the counter to one
loop:
beq     $t1, $a1, exit    #exit if we reach the end of the array
addi    $a0, $a0, 4       #increment the pointer by one word
addi    $t1, $t1, 1       #increment the loop counter
lw      $t2, 0($a0)       #store the next array value into $t2
slt     $t3, $t0, $t2
beq     $t3, $0, end_if
add     $t0, $0, $t2      #found a new maximum,store it in t0
end_if:
j       loop              #repeat the loop
exit:
```

The equivalent C statement is the following (one version of many possible ways):

```
void max ( int array[ ], int end ) {

  // Parameters passed are stored in registers $a0 and $a1.

  int maxNumber = array[0]; // Load the first array value into $t0.
  int i = 1; // Initialize the counter (i.e. register $t1) to one.
  int temp; // Temporary variable (i.e. $t2).

  // Exit while loop if we reach the end of array.
  while ( i != end ) {

    i = i + 1; // Increment the loop counter by 1.
    temp = array[ i - 1 ]; // Store the next array value.
    if ( maxNumber < temp )
      maxNumber = temp; // Found a new maximum number.

  } // end while loop.
} // end function.
```

In essence, the previous MISP code is finding the largest value in an array of integers.

4. For the MIPS assembly instructions below, what is the corresponding C statement?

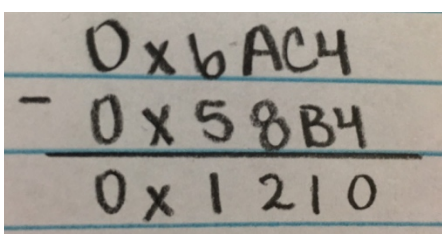
```
lw      $t0 , 4($gp)   #fetch N
mult    $t0 , $t0 , $t0
lw      $t1 , 4($gp)   #fetch N
ori     $t2, $zero, 3
mult    $t1, $t1, $t2
add     $t2, $t0, $t1
sw      $t2, 0($gp)
```

The corresponding C statement is the following (one version of many possible ways):

```
// Assume global static variables are stored somewhere
// in memory "right next to each other".
static int i; // 0($gp)
static int N; // 4($gp)
.
.
.
i = N*N + N*3; // Actual C implementation of MIPS code.
```

5. What is 6AC4 – 58B4 when these values represent **unsigned** 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

Method 1: Direct Subtraction



ANSWER: 0x6AC4 – 0x58B4 = 0x1210

Method 2: "Long Way"

First convert 6AC4 to decimal:

$$6AC4 = 6 * 16^3 + 10 * 16^2 + 12 * 16^1 + 4 * 16^0 = 24576 + 2560 + 192 + 4 = 27332$$

Now convert 58B4 to decimal:

$$58B4 = 5 * 16^3 + 8 * 16^2 + 11 * 16^1 + 4 * 16^0 = 20480 + 2048 + 176 + 4 = 22708$$

We note that $27332 - 22708 = 4624$

We now have to convert 4624 into hexadecimal.

$$\frac{4624}{16} = 289 \; with \; reminder \; 0$$

$$\frac{289}{16} = 18 \; with \; reminder \; 1$$

$$\frac{18}{16} = 1 \; with \; reminder \; 2$$

$$\frac{1}{16} = 0 \; with \; reminder \; 1$$

ANSWER: 0x6AC4 – 0x58B4 = 0x1210

6. What decimal number does the bit pattern 0x8CF00000 represent if it is a two's complement integer? An unsigned integer?

- Two's Complement Integer

If the number is a two's complement integer, we can compute the two's complement to find the equivalent positive number.

0x8CF00000 = 0b10001100111100000000000000000000

We now can use the equation found on the slides:

$$1 * -2^{31} + 1 * 2^{27} + 1 * 2^{26} + 1 * 2^{23} + 1 * 2^{22} + 1 * 2^{21} + 1 * 2^{20}$$
$$= -2147483648 + 134217728 + 67108864 + 8388608 + 4194304 + 2097152 + 1048576$$
$$= -1930428416$$

ANSWER: The corresponding two's complement integer is -1930428416

- Unsigned Integer

If the number is an unsigned integer, we can directly compute its equivalent decimal value:

$$0x8CF00000 = 8 * 16^7 + 12 * 16^6 + 15 * 16^5 = 2147483648 + 201326592 + 15728640$$
$$= 2364538880$$

ANSWER: The corresponding unsigned integer is 2364538880

7. Assume the following register contents: \$t0 = 0xAAAAAAAA, \$t1 = 0x12345678
For the register values shown above, what is the value of \$t2 for the following sequence of instructions?

```
Sll  $t2, $t0, 44
Or  $t2, $t2, $t1
```

First, we note that \$t0 contains the hexadecimal number 0xAAAAAAAA, which equals to 0b1010-1010-1010-1010-1010-1010-1010-1010 in binary. However, if we shift this number a total of 44 times to the left, we get 0x00000000. *(NOTE: These particular registers can only store 32 bits. If they stored 64 bits, then we would have 0xAAAAAAAA00000000 after 44 shifts to the left. However, this is not the case for this particular example.)* That is, \$t2 contains 0x00000000 after the first line executes. When the second line executes, we do a logical "or" between 0x00000000 (register \$t2) and 0x12345678 (register \$t1). Any binary number that is logically "or" with zero will return the same original number. That is:

```
0001 0010 0011 0100 0101 0110 0111 1000
0000 0000 0000 0000 0000 0000 0000 0000
_______________________________________
0001 0010 0011 0100 0101 0110 0111 1000 = 0x12345678
```

ANSWER: \$t2 holds 0x12345678 after the sequence of instructions is executed.

8. Assume \$t0 holds the value 0x80101000. What is the value of \$t2 after the following instructions?

```
        slt   $t2, $0,  $t0
        bne   $t2, $0,  ELSE
        j     DONE
ELSE:   addi  $t2, $t2, 2
DONE:
```

We note that \$t0 = 0x80101000. We also assume that \$0 represents the \$zero register, which contains the number 0x00000000. The first line checks if the \$zero register contains a number less than register \$t0. Is important to note that slt checks for *signed* numbers. We also note that 0x80101000 = 0b10000000000100000001000000000000. Because we are representing this as a signed number, 0x80101000 is negative since its most significant bit is 1. Since 0 < *any negative number* is never true, \$t2 holds 0x00000000 after the first line is executed. The second line checks if the numbers 0x00000000 and 0x00000000 are not equal. That is, "if zero and zero are not equal, branch to else." Since both \$t2 and \$0 contain the number zero, the conditional branch fails and falls through, . We then jump to the DONE label (skipping the fourth line), thus never modifying \$t2 after line 1.

ANSWER: \$t2 holds 0x00000000 (i.e. the number zero) after the previous instructions are executed.