Keegan Smith
Computer Architecture and Design
HW1
10/10/2023

1) The results of the benchmark named P running on CPU A has an IC (instruction count) of 2.389*10^12 and an execution time of 750 seconds.
   a. Find the CPI if the clock cycle time is 0.333 ns.

   $$CPU\ time = IC * CPI * Clock\ Cycle\ time$$

   $$CPI = \frac{CPU\ time}{IC * Clock\ cycle\ time} = \frac{750}{(2.389 * 10^{12}) * 0.333ns} = 0.95133 \frac{cycles}{instruction}$$

   b. Find the increase in CPU time if the number of instructions of the benchmark is increased 2% without affecting the CPI.

   $CPU\ Time = IC * CPI * Clock\ Cycle\ Time$
   $CPU\ Time = (1.02)IC * CPI * Clock\ Cycle\ Time$
   $CPU\ Time = (1.02 * 2.389 * 10^{12}) * 0.95133 * 0.333\ ns = 771.9545 = 772\ seconds$
   So, the increase in the number of instructions resulted in a 103% increase in CPU time.

   c. Find the increase in the CPU time if the number of instructions of the benchmark is increased by 10% and the CPI is increased by 20%.

   $CPU\ Time = (10\%)IC * (20\%)CPI * Clock\ Cycle\ Time$
   $CPU\ Time = (1.10 * 2.389 * 10^{12}) * (1.20 * 0.95133) * 0.333\ ns = 999\ seconds$
   The increase in instructions and CPI generated a 133% increase in the CPU time from the initial benchmark.

   d. Suppose that we are developing a CPU B with a 4GHz clock rate. We have also added some additional instructions to the instruction set so that the number of instructions has been increased by 5%. The execution time is reduced to 725ns. Find the new CPI.

   $$Clock\ Cycle\ Time = \frac{1}{Clock\ Rate} = \frac{1}{4 * 10^9} = 0.25 * 10^{-9}\ seconds$$

   $$CPI = \frac{CPU\ time}{IC * Clock\ cycle\ time} = \frac{725}{1.05(2.389 * 10^{12}) * 0.25ns} = 1.156 \frac{cycles}{instruction}$$

e. This CPI value is larger than obtained in 1.a as the clock rate was increased from 3GHz to 4GHz. Determine whether the increase in the CPI is similar to that of the clock rate. If they are dissimilar, why?

The increase in CPI is similar to that of the clock rate because of the linearity of the CPI equation when a single variable is changing. If more than one variable was being adjusted, then the relationship may not be so similar.

f. By how much has the CPU time been reduced?

The CPU time can be reduced by reducing the instruction count, cycles per instruction, or decreasing the clock rate. Since CPU Time = (IC * CPI) / clock rate, making IC or CPI smaller would result in a smaller number. Once could also slow down the CPU speed. This may seem backwards, but the CPU time formula, the clock rate in seconds is given so, the larger the clock frequency, the smaller the clock rate will be and the larger the CPU time will be because of the larger denominator.

2) Translation between C statements and MIPS assembly instructions.

a. Assuming variables f, g, h and I are given as 32-bit integers, what are the corresponding MIPS assembly instructions for the following C program? Use a minimal number of instructions. $f = g + (h - 2)$

```
addi    $t0, h, -2 # temp t0 = h - 2
add     $t0, $t0, g # temp t0 = t0 + g
```

b. What is the corresponding C statement for the following MIPS assembly instructions?

Add f, g, h
Add f, i, f

```
f = g + h;
f += i;
```

c. What is the corresponding MIPS assembly code for the C statement below? Assume that the variables f, g, h, i, and j are assigned to registers \$s0, \$s1, …, \$s4, respectively. Assume also that the base address of the array A and B are stored in registers \$s6 and \$s7, respectively. B[6] = A[i-j];

```
#variable declaration
sw      $s0, f
sw      $s1, g
sw      $s2, h
sw      $s3, i
sw      $s4, j

# array starting addresses
sa      $s6, A
sa      s$7, B

# do the offset calculation
negu    $t0, $s4        # $t0 = -j
add     $t1, $s3, $t0   # $t0 = i + $t0 == i - j
sll     $t1, $t1, 2     # perform 4*(i-j) to get offset value

# save the value of A[i-j] to $t3
lw      $t3, $t1($s6)

# do the offset calculation of B[6]
li      $t4, 6              #load 6 into $t4
sll     $t4, $t4, 2         # $t4 = $t4 * 4

# save A[i-j] to B[6]
sw      $t3, $t4($s7)

```

d. What is the corresponding C statement for the following MIPS assembly instructions? Assume that the variables and base address of the arrays are the same as 2.c.

```
sll $t0, $s0, 2 # $t0 = f *4
add $t0, $s6, $t0 # $t0 = &A[f]
sll $t1, $s1, 2 # $t1 = g*4
add $t1, $s7, $t1 # $t1 = &B[g]
lw $s0, 0($t0) # f = A[f]
addi $t2, $t0, 4
lw $t0, 0($t2)
add $t0, $t0, $s0
sw $t0, 0($t1)
```

```
B[g] = A[f + 1] + A[f];
f = A[f];
```

3) The table below shows 32-bit values of an array stored in memory.

Address	Data
24	2
28	4
32	3
36	6
40	1

a. For the memory location in the table above, write C code to sort the data from lowest to highest; in other words, placing the lowest value in the smallest memory location.

```
C: > Users > Keegan > Desktop > lowell > Fall 2023 > CAaD > C test.c > clangd > selectionSort
#include <stdio.h>
// Code sourced from online @: https://www.geeksforgeeks.org/c-program-to-sort-an-array-in-ascending-order/


void swap(int* xp, int* yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// Function to perform Selection Sort
void selectionSort(int arr[], int n)
{
    int i, j, min_idx;

    // One by one move boundary of
    // unsorted subarray
    for (i = 0; i < n - 1; i++)
    {
        // Find the minimum element in
        // unsorted array
        min_idx = i;

        for (j = i + 1; j < n; j++)
        {
            if (arr[j] < arr[min_idx])
            {
                min_idx = j;
            }
        }
        // Swap the found minimum element
        // with the first element
        swap(xp: &arr[min_idx], yp: &arr[i]);
    }
}

// Function to print an array
void printArray(int arr[], int size)
{
    int i;
    for (i = 0; i < size; i++)
        printf(Format: "%d ", arr[i]);
    printf(Format: "\n");
}

// Driver code
int main()
{
    int arr[] = { [0]=2, [1]=4, [2]=3, [3]=6, [4]=1 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printf(Format: "Original array: \n");
    printArray(arr, size: n);

    selectionSort(arr, n);
    printf(Format: "\nSorted array in Ascending order: \n");
    printArray(arr, size: n);

    return 0;
}
```

b. Write the same sort code using MIPS instructions. Use a minimum number of MIPS instructions. Assume that the base address of Array is stored in register $s6.

```
#code sourced from online @: https://stackoverflow.com/questions/19212544/sorting-array-in-mips-assembly
#then changed to HW parameters

.text

.globl main

main:
    la  $t0, Array              # Copy the base address of your array into $t1
    add $t0, $t0, 20            # Length of array calculation, 4 bytes per int * 5 ints = 20 bytes

outterLoop:                     # Used to determine when we are done iterating over the Array
    add $t1, $0, $0             # $t1 holds a flag to determine when the list is sorted
    la  $s6, Array              # Set $s6 to the base address of the Array

innerLoop:                      # The inner loop will iterate over the Array checking if a swap is needed
    lw  $t2, 0($s6)             # sets $t0 to the current element in array
    lw  $t3, 4($s6)             # sets $t1 to the next element in array
    slt $t5, $t2, $t3           # $t5 = 1 if $t0 < $t1
    beq $t5, $0, continue       # if $t5 = 1, then swap them
    add $t1, $0, 1              # if we need to swap, we need to check the list again
    sw  $t2, 4($s6)             # store the greater numbers contents in the higher position in array (swap)
    sw  $t3, 0($s6)             # store the lesser numbers contents in the lower position in array (swap)

continue:
    addi $s6, $s6, 4            # advance the array to start at the next location from last time
    bne  $s6, $t0, innerLoop    # If $s6 != the end of Array, jump back to innerLoop
    bne  $t1, $0, outterLoop    # $t1 = 1, another pass is needed, jump back to outterLoop

.data

Array:  .word   2, 4, 3, 6, 1
```