Performance Analysis of MIPS Assembly Code Calculating Dot Product

I. Introduction

The scope of this project is to calculate the processor performance and energy consumption of two different processors as they compute the dot product of two vectors. The processors are described in the sections below.

Processor A. Processor A uses a subset of the MIPS instruction set architecture (ISA). The key difference in the ISA of Processor A and the MIPS ISA is that there is no multiplication instruction. The instructions used are as follows: *lw*, *add*, *la*, *li*, *add*, *addi*, *blt*. Processor A is assumed to have a clock frequency of 1GHz. Like MIPS, it has a 4-byte word size.

Processor B. Processor B uses a larger subset of the MIPS ISA. This ISA includes a multiplication instruction, in addition to Processor A's instructions listed above. Like Processor A, Processor B is assumed to have a clock frequency of 1GHz. It also uses a 4-byte word size.

II. Data

Both processors are tested using four vector pairs of varying lengths: 5, 10, 20, and 50. The last vector pair, of length 50, is used for testing the code and performance evaluation. The vector pair, *A* and *B*, and vector length, *n*, are shown in Table 1 below.

Table 1: Dot product data.

A	2, 2, 3, 4, 5, 2, 2, 3, 4, 5, 3, 7, 8, 9, 10, 2, 2, 3, 4, 5, 2, 2, 3, 4, 5, 2, 2, 3, 4, 5, 3, 7, 8, 9, 10, 2, 2, 3, 4, 5, 8, 8,
	8, 8, 8, 8, 8, 8, 8, 8
B	3, 7, 8, 9, 10, 2, 2, 3, 4, 5, 3, 7, 8, 9, 10, 2, 2, 3, 4, 5, 2, 2, 3, 4, 5, 2, 2, 3, 4, 5, 3, 7, 8, 9, 10, 2, 2, 3, 4, 5, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8
n	50

III. Code

Two programs were written for calculating the dot product of two vectors. Program A. Program A was written for Processor A, and Program B was written for Processor B. First, the similar portions of the programs shall be discussed, and then the differences caused by the differing ISA shall be discussed.

Workload. The workload of these two processors is computing a dot product. A dot product is defined as the sum of the products of two vectors of equal lengths at each index.

Initialization. The program begins by loading the vector length and word size into temporary registers. The variable to hold the result of the dot product is also created. The dot product is initialized to zero and will be increased as each product of the vectors is added to it. The dot product is computed using a loop, so a temporary register is designated to hold an iterator variable that will increment by one as each product is calculated. The loop will run for the vector length. The last step of initialization is to load in the addresses of the two input vectors. Figure 1 below shows the code for

the initialization steps.

```
lw $t0, n  # vector length

# sum: holds the dot product
add $t1, $zero, $zero
# i: dot product loop iterator
add $t2, $zero, $zero
lw $t7, word_length

# addresses pointers of vectors A, B
la $t3, A
la $t4, B
```

Figure 1: Initialization steps.

Dot Product Loop. A loop is needed to calculate the dot product. After every loop iteration, the value of the loop iterator is compared to the vector length. When the iterator is not less than the vector length, the loop is allowed to exit. Inside the loop, the first step is to load the vector at each index into a register so that it can be calculated. The *load word* instruction is used with the offset set at 0 and the base register set as the address of the vector at index i. At the end of the loop the address is incremented by the word size to load the vector at the next index. The iterator variable is also incremented. Finally, inside the loop, the vectors at i are multiplied and their product added to the sum. The multiplication implementation is shown in the following section. Figure 2 below shows the dot product loop, excluding the multiplication.

```
# loop over every index of the vectors
DOT_PRODUCT_LOOP:
    lw $s1, 0($t3)
    lw $s2, 0($t4)

    # ** multiplication goes here **

    # add product to sum
    add $t1, $t1, $t6

    addi $t2, $t2, 1  # ++i
    add $t3, $t3, $t7  # address + 4
    add $t4, $t4, $t7  # address + 4

    # i < vector length
    blt $t2, $t0, DOT_PRODUCT_LOOP
```

Figure 2: Dot product loop.

Multiplication. As stated in the previous section, multiplication is implemented differently for the different processors. Processor A does not include a multiplication instruction, so a loop is implemented to add *A[i]* to itself *B[i]* times. Another iterator is implemented to exit the loop when it is greater than or equal to *B[i]*. The product is stored in a temporary register. Figure 3 below shows multiplication for Processor A.

```
    # multiply vectors at i

    # j: multiplier loop iterator
    li $t5, 0
    li $t6, 0  # product

    MULTIPLIER_LOOP:
        add $t6, $t6, $s1

        addi $t5, $t5, 1  # ++j
        # j < b[i]
        blt $t5, $s2, MULTIPLIER_LOOP
```

Figure 3: Multiplication for Processor A.

Implementing multiplication in Processor B is more simple. It only requires one multiplication instruction. The instruction is shown in Figure 4 below.

```
# multiply vectors at i
mul $t6, $s1, $s2
```

Figure 4: Multiplication for Processor B.

Testing. The result of the dot product from

the vectors in Table 1 is 1666. For both programs, the result is stored in temporary register *$t1*. The value of *$t1* after program execution is shown as 0x00000682, which is 1666 in hexadecimal. Figure 5 shows the assembled code and register contents after running the program.

IV. Evaluation

The assembly code for both processors was assembled and run using MIPS Assembler and Runtime Simulator (MARS) 4.5. The input data used for evaluation was the vector pair described in Table 1. Below, the CPU performance and energy consumption of the two processors is described. The MIPS/mW for each processor is also calculated.

CPU Performance.

CPU performance can be calculated with the average number of clock cycles per instruction (CPI) in a workload. CPI is the number of each instruction type times the clock cycles per instruction of that type, added together and divided by the number of instructions. For this purpose, it is assumed R-type instructions require an average of 3 clocks, I-type instructions require an average of 2 clocks, and J-type instructions require an average of 5 clocks. The MIPS Instruction Counter tool provides the number of each type of instructions. CPI is calculated with the following equation:

```
CPI = [3(number of R-type
instructions) + 2(number of I-type
instructions) + 5(number of J-type
instructions)] / total number of
instructions
```

Equation 1: Clocks per instruction.

Processor A CPI. Processor A uses 1595 instructions, according to the MIPS Instruction Counter tool. It uses 744 R-type instructions, 850 I-type instructions, and 0 J-type instructions. The result of the instruction counter is shown in Figure 6.

```
CPI = [3(744) + 2(850) + 5(0)] / 1595
CPI = 3932 / 1595
CPI = 2.47
```

Figure 7: Processor A CPI calculation.

As shown in Figure 7, Processor A uses 2.47 clocks per instruction.

Processor B CPI. Processor B uses 461 instructions, according to the MIPS Instruction Counter tool. It uses 252 R-type instructions, 208 I-type instructions, and 0 J-type instructions. The result of the instruction counter is shown in Figure 8.

```
CPI = [3(461) + 2(208) + 5(0)] / 461
CPI = 1799 / 461
CPI = 3.90
```

Figure 9: Processor B CPI calculation.

As shown in Figure 9, Processor B uses 3.9 clocks per instruction.

Table 2. Comparison of instruction count for both processors.

	Total	**R-type**	**I-type**	**J-type**
A	1595	744	850	0
B	461	252	208	0

Energy Consumption.

The energy consumed by both processors is calculated assuming an energy consumption per instruction as described below in Table 3 below.

Table 3: Energy consumption of instruction by type.

ALU	2 fj

Jump	5 fj
Branch	5 fj
Memory	20 fj
Other	4 fj

Energy consumption is calculated with the following equation:

```
Total energy consumption =
2(number of ALU instructions) +
5(number of jump instructions) +
5(number of branch instructions) +
20(number of memory instructions) +
4(number of other-type instructions)
```

Equation 2: Energy consumption

The MIPS Instruction Statistics tool shows the number of each instruction as shown in Figure 10 and Figure 11. The instruction number for both processors is also shown in Table 4 below.

Table 4: Instruction number by type for each processor.

	A	**B**
ALU	850	208
Jump	0	0
Branch	321	50
Memory	102	102
Other	321	100

Processor A Energy Consumption. Energy consumption is calculated in Figure 12 below:

```
Total energy consumption = 2(850) +
5(0) + 5(321) + 20(102) + 4(321)

Total energy consumption = 1700 + 0 +
1605 + 2040 + 1284

Total energy consumption = 6629 fj
```

Figure 12: Processor A energy consumption calculation.

As shown above, the total energy consumption for Processor A is 6629 femtojoules.

Processor B Energy Consumption. Energy consumption is calculated in Figure 13 below:

```
Total energy consumption = 2(208) +
5(0) + 5(50) + 20(102) + 4(100)

Total energy consumption = 416 + 0 +
250 + 2040 + 400

Total energy consumption = 3106 fj
```

Figure 13: Processor B energy consumption calculation.

As shown above, the total energy consumption for Processor B is 3106 femtojoules.

MIPS/mW

The MIPS per mW for a processor are calculated as simply the MIPS divided by the mW. MIPS (Million Instructions per Second) is the number of million instructions per second a processor can execute. It is calculated with the equation below:

```
MIPS = (Clock rate)/(CPI x 10^6)
```

Equation 3: MIPS

Power in milliwatts is the amount of energy over the execution time, shown in

Equation 4 below:

```
Power (mW) = (Energy/Execution Time) x
10^3
```

Equation 4: Power (mW)

Finally, execution time is calculated as described below. It requires the clock period, which can be calculated from the clock frequency, also shown below:

```
Period = 1/Frequency
```

Equation 6: Clock period

```
Execution Time = Clocks x Period
```

Equation 7: Execution time

Processor A. The calculations for MIPS/mW for Processor A are shown in Figure 14 below:

```
MIPS = (1 GHz)/(2.47 x 10^6)
MIPS = 2.47 x 10^15

Period = 1/1 GHz
Period = 1 nanosecond

Execution Time = 3932 clocks x 1
nanosecond
Execution Time = 3932 nanoseconds

Power (mW) = (6629 fj/3932
nanoseconds) x 10^3
Power (mW) = (1.686 x 10^-6) x 10^3
Power (mW) = 1.686 x 10^-3

MIPS/mW = (2.47 x 10^15)/(1.686 x
10^-3)
MIPS/mW = 1.465 x 10^18
```

Figure 14: Processor A MIPS/mW calculations.

Processor A computes 2.47×10^{15} millions of instructions per second for this workload. The workload consumes 1.686×10^{-3} milliwatts. The energy efficiency of this processor for this workload is 1.465×10^{18} MIPS per milliwatt.

Processor B. The calculations for MIPS/mW for Processor B are shown in Figure 15 below:

```
MIPS = (1 GHz)/(3.9 x 10^6)
MIPS = 3.9 x 10^15

Period = 1/1 GHz
Period = 1 nanosecond

Execution Time = 1799 clocks x 1
nanosecond
Execution Time = 1799 nanoseconds

Power (mW) = (3106 fj/1799
nanoseconds) x 10^3
Power (mW) = (1.726 x 10^-6) x 10^3
Power (mW) = 1.726 x 10^-3

MIPS/mw = (3.9 x 10^15)/(1.726 x
10^-3)
MIPS/mW = 2.259 x 10^18
```

Figure 15: Processor B MIPS/mW calculations.

Processor B computes 3.9×10^{15} millions of instructions per second for this workload. The workload consumes 1.726×10^{-3} milliwatts. The energy efficiency of this process for this workload is 2.259×10^{18} MIPS per milliwatt.

V. Conclusion

The different instruction set architectures impact both the performance and energy consumption of the processor. Processor B, at 3.9 CPI, is slower than Processor A at 2.47 CPI. It uses less instructions, but the instructions require more clocks because they are more complex. Specifically, the multiplication instruction will require more clocks than an addition instruction, which can be used

repeatedly for multiplication.

Processor A consumes over twice the energy that Processor B consumes. This is because the instruction count of Processor A is over three times the instruction count of Processor B. More instructions use more energy.

Processor B has a higher MIPS per milliwatt than Processor A, meaning it is more energy efficient for this particular workload. Table 5 below compares the MIPS/mW results of Processor A and Processor B to show energy efficiency.

Table 5: Comparison of MIPS, power, and MIPS/mW for Processors A and B.

	MIPS	Power (mW)	MIPS/mW
A	2.47×10^{15}	1.686×10^{-3}	1.465×10^{18}
B	3.9×10^{15}	1.726×10^{-3}	2.259×10^{18}

Processor B is slower but more energy efficient than Processor A for the workload of computing a dot product for two vectors of size 50.

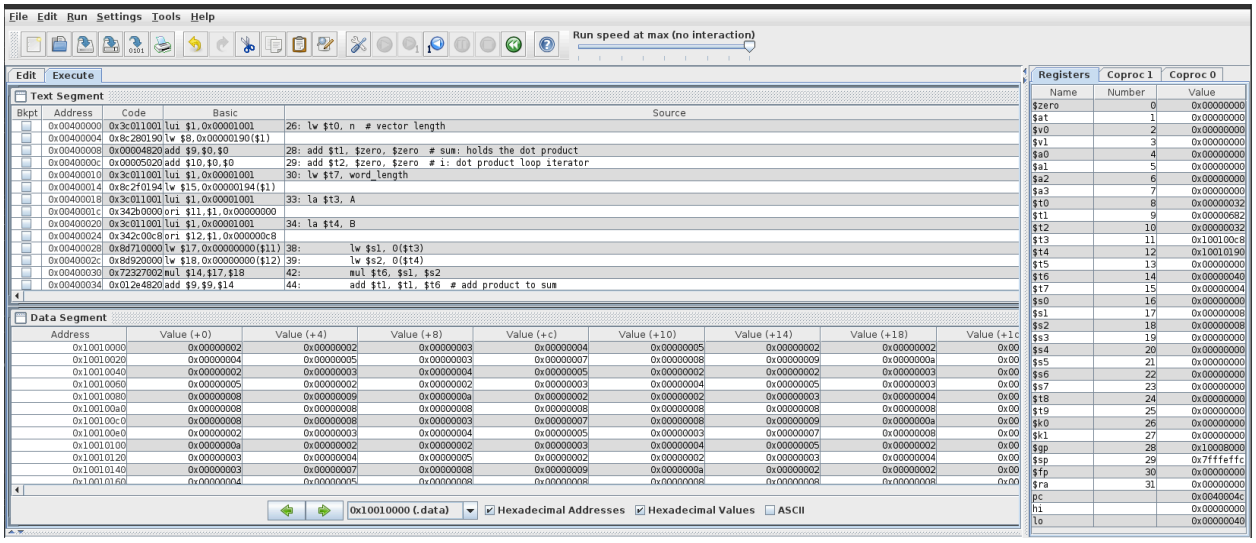

Figure 5. Output results

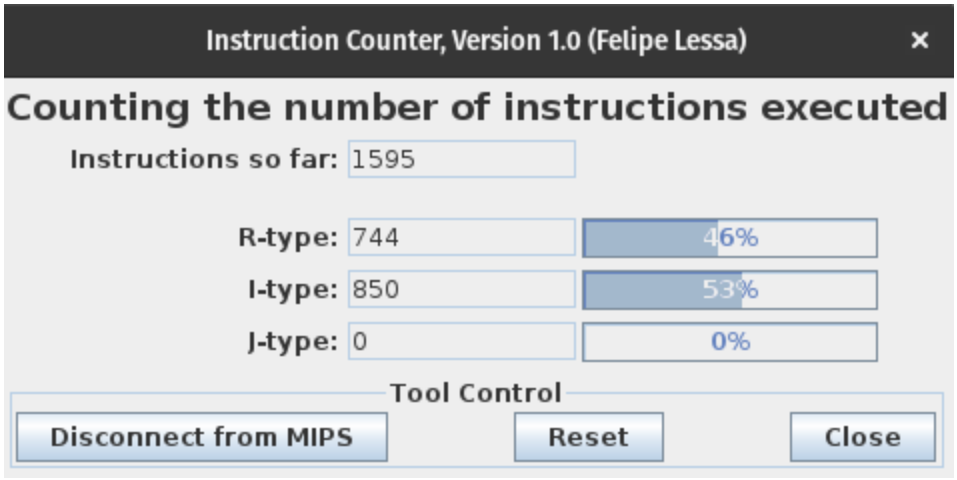

Figure 6. Instruction counter output for Processor A.

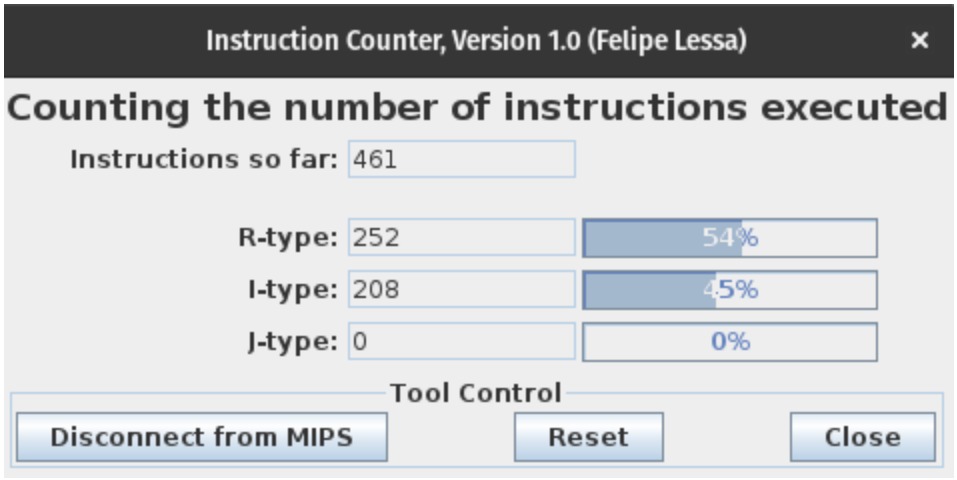

Figure 8. Instruction counter output for Processor B.

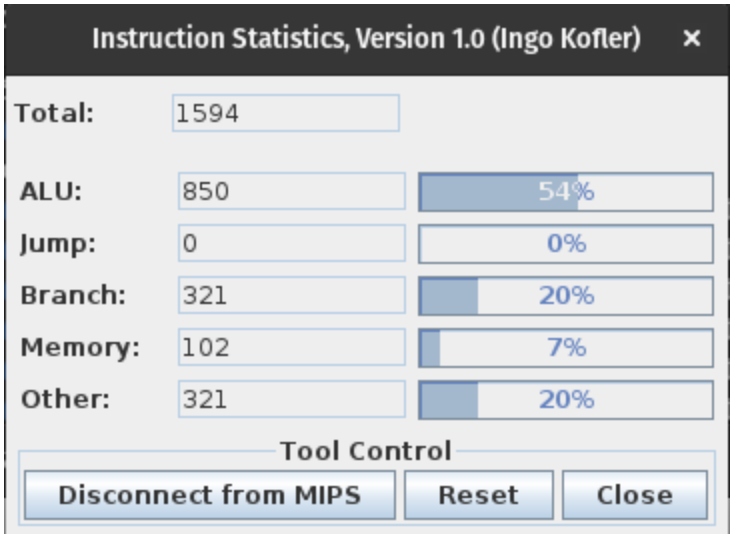

Figure 10. Instruction statistics output for Processor A.

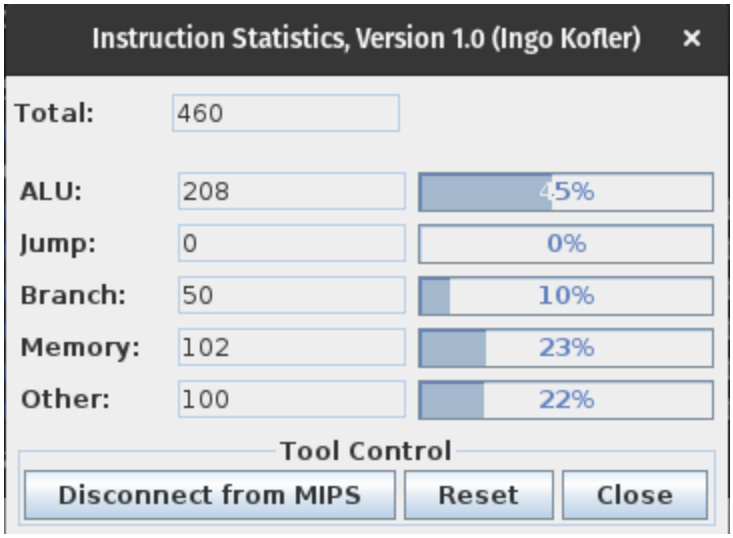

Figure 11. Instruction Statistics for Processor B.