Power Consumption of Instruction Encodings on Cortex-M4

A report submitted for the course
COMP2300 (Advanced Studies Extension)

Zixian Cai
Research School of Computer Science, Australian National University

June 14, 2018

Abstract

Energy efficiency is increasingly important with wider use of battery-powered devices. There are many factors involved in the power consumption of instructions, such as encodings. The energy implication of these factors is not clear, and needs to be evaluated.

The main contribution of this work is the characterization of power consumption of some instruction encodings of ARM v7-M on Cortex-M4 MCU `STM32L476VGT6`. I also designed and implemented a self-contained power measurement infrastructure on `32L476GDISCOVERY`.

I found that the encodings of instructions could affect the energy consumption of the MCU.

The results of this work can make application programmers and hardware vendors be aware of the energy characteristics of programs and devices. Future work of a more detailed model could allow us to predict the energy consumption when designing new systems.

1 Introduction

1.1 Problem Statement

Computers, from IoT devices to supercomputers, are widely deployed nowadays. Despite the vastly different hardware they are running, the power consumption of these devices has always been a main concern. Being power efficient, mobile devices can have longer battery life, and millions of dollars can be saved for electricity bill of server farms.

On commonly used ISAs, such as ARM v7-M, there are multiple factors that could potentially affect the energy consumption of a device. For example, different encodings of the same instruction may lead to different power consumption, as they have different code sizes and map to different hardware capabilities. On the one hand, wider instructions can lead to pressure on the cache and memory bus, and potentially consumes more energy. On the other hand, these wide instructions provide more functionality, which can improve the hardware utilization.

The power implication of different factors in using instructions is not clear, and experiments need to be carried out.

1.2 Contributions

I first designed and implemented a self-contained power measurement framework on `32L476GDISCOVERY`. This framework does not rely on extra hardware and adapter, except for the board to be measured itself. It also allows automated collection of data points at scale.

Through the use of this framework, I evaluated the power consumption of different encodings of `ADD`, `SUB` and `RSB` instructions. I also demonstrated that this approach could be used on non-arithmetic instructions, such as `LDR` and `STR`.

1.3 Outline

Section 2 describes the design and implementation of the power consumption measurement framework. Then, Section 3 presents the results gathered via the framework, and discusses potential factors that could affect the energy consumption of instructions. Finally, Section 4 summarizes the work, and points out potential future extension to this project. The artefact can be found at https://gitlab.anu.edu.au/u5937495/comp2300-ase.

2 Design and Implementation

2.1 Measurement Kernels

The goal of this work is to find out the power implication of different instructions. Due to my time constraints, only a small range of operations are evaluated.

Among these instructions, `ADD` and `SUB` are particularly interesting. They have multiple different encodings that support difference range of operations. For example, the encoding that uses 4 bits, rather than 3 bits, to specify register operands can access all 16 of the registers. In addition, they support on-the-fly bit-shifting to operands via the barrel shifter, which requires wider encodings.

Beside the arithmetic instructions mentioned above, I also evaluated memory operations, such as load/store (register), for the sake of completeness.

For each measurement kernel, the instruction of interest needs to execute for a long period of time so that the the energy consumption will stabilize and converge. In order to minimize the impact of unrelated code, a script is used to repeat a sequence of instructions based on the description of the kernel (see Fig. 1(a) and Fig. 1(c)). It has the advantage over loops that we are not measuring the overall energy cost of target instructions and branching (see Fig. 1(b) for an example).

Note that the assembly instructions and mnemonics are used in Fig. 1 only for illustration purposes. For the widely used GNU Assembler, it does not guaranteed whether a particular encoding would be used to implement an assembly instruction. Therefore, in the actual kernels used in this work, instructions are explicitly transcoded to binary forms, so that we have precise control over the encoding (see Fig. 2).

```
{
    "prologue": "ldr r1, [r0]\nmov r2, #2\n",
    "body": "add r1, r2\n",
    "epilogue": "str r1, [r0]\n",
    "size": 100
}
```

(a) Description of a Kernel

```
benchmark:
    ldr r1, [r0]
    mov r2, #2
    mov r3, #0
loop:
    add r1, r2
    add r3, #1
    cmp r3, #100
    blt loop
    str r1, [r0]
    bx lr
```

(b) Implementation using loop

```
benchmark:
    ldr r1, [r0]
    mov r2, #2
    add r1, r2
    add r1, r2
    ...97 more times...
    add r1, r2
    str r1, [r0]
    bx lr
```

(c) Generated by script

Figure 1: Measurement kernel: an example

2.2 Power Measurement

It is difficult to find out the exact energy consumption of an instruction. Therefore, the current of the entire MCU is measured as the proxy metric.

The `STM32L476VGT6` MCU[1] is a part of the STM32L4 series that target ultra-low-power embedded devices. These embedded devices are sensitive to power consumption, and thus makes the MCU a natural target for my experiments.

`32L476GDISCOVERY`[2] is a development kit that hosts an `STM32L476VGT6` MCU. It provides an on-board MCU ammeter with 4 ranges and auto calibration. This allows easily accessible measurements without additional hardware, and therefore makes it an ideal choice as my hardware platform. The schematic

[1]http://www.st.com/en/microcontrollers/stm32l476vg.html
[2]http://www.st.com/en/evaluation-tools/32l476gdiscovery.html

```
{
    "prologue": "ldr r1, [r0]\nmov r0, #1\n",
    "body": ".hword 0b0001100000001001\n", // add r1, r0
    "epilogue": "",
    "size": 100
}
```

Figure 2: Measurement kernel with explicit encoding

of the measurement circuit can be found in the ST manual UM1879[3] with title IDD measurement / Multi-Function eXpander (MFX).

The BSP library provided with the board supplies various functions for interacting with the ammeter. This library controls the MFX to translate the voltage signals into current readings in the unit of 10nA. The usage of the library can be found at https://gitlab.anu.edu.au/u5937495/comp2300-ase/blob/master/src/idd.c.

2.3 Data collection

Once the current reading is gathered from the on-board ammeter, the USB OTG port is used to transmit the data to a host computer. The USB OTG port can be configured as a USB communications device class (CDC) device, whose support presents in most modern operating systems. On these supported platforms, `32L476GDISCOVERY` is exposed to the host computer as a virtual COM port (VCP) (see Fig. 3). Then, standard tools like `screen` can be used to read the current readings from the serial device. This infrastructure enables automated data collection, which allows me to collect data points at a larger scales. STM32CubeMX[4] is used to generate boilerplate code for USB CDC.

Other alternative approaches are also evaluated, but they are not used.

LCD screen `32L476GDISCOVERY` has a on-board six-digit LCD screen, which can be used to display current readings. However, it requires manual data logging, which is not feasible when we want to gather a large amount of data points. In addition, the current reading does not fit in the screen, and therefore either the scrolling display needs to be used or the data need to rounded. Either way, it increases the difficulty of data collection.

ST-LINK VCP The ST-LINK MCU on `32L476GDISCOVERY` is connected to one of the USART RX/TX ports of the main MCU. Like the USB OTG approach, the ST-LINK protocol also supports VCP. In addition, the communication goes through the same cable for flashing, which makes the connection simpler. However, the ST-LINK VCP requires a special driver, which to my best knowledge, is only provided on Windows. This impedes portability.

3 Results

The results obtained via the measurement infrastructure discussed in Section 2 will be presented and discussed in this section. The kernels can be found

[3]http://www.st.com/resource/en/user_manual/dm00172179.pdf

[4]http://www.st.com/en/development-tools/stm32cubemx.html

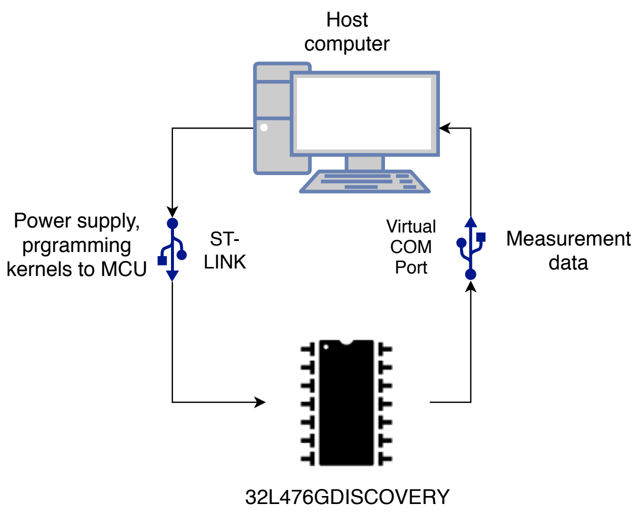

Figure 3: USB OTG VCP: organization of hardware

at https://gitlab.anu.edu.au/u5937495/comp2300-ase/tree/master/src/kernels, and the original data can be found at https://gitlab.anu.edu.au/u5937495/comp2300-ase/tree/master/results.

3.1 Power Consumption of Instructions

Different kernels are interleaved when conducting experiments. For example, instead of running kernels in the order of `AABBCC`, the order `ABCABC` is used. This is to avoid one kernels being skewed by short term background noises. The results are summarised in Table 1[5].

Kernel	Addressable registers	Barrel shifter	Width	Mean energy consumption (10nA)
`add_reg_T1`	8	No	16	1414594.3
`add_reg_T2`	16 for Rm, 8 for Rdn	No	16	1411220.85
`add_reg_T3_lsl0`	16	Yes	32	1402551.8
`ldr_reg_T1`	8	No	16	1625625.45
`str_reg_T1`	8	No	16	1664383.65
`sub_reg_T1`	8	No	16	1473407.9
`sub_reg_T2`	16	Yes	32	1458067.6
`rsb_reg_T1`	16	Yes	32	1470169.55

Table 1: Characteristics of instructions tested

[5]Instructions usually have registers as their operands. However, the addressable registers are different from encodings to encodings and usually depend on the number of bits used to specify a register.

3.2 Statistical Analysis

In order to find out whether the difference in instructions leads to different energy consumption, analysis needs to be performed. Independent two sample t-test is a commonly applied statistical hypothesis test. In order to use it, however, several assumptions need to be satisfied.

The current readings gathered can be assumed to be independently sampled, as the experiments are interleaved (see Section 3.1). Furthermore, the assumption are made that the current readings follow normal distribution.

In the following sections, t-test is used to discuss the results obtained, and both the sample size and p-value will be reported.

3.3 The Encodings of Addition Instructions

The MCU used in this work supports the Thumb-2 instruction set, which is an extension to the 16-bit Thumb instruction set. It provides 32-bit instructions that cover some functionalities of the ARM instruction set, such as the barrel shifter. The `ADD` (register) instruction has 3 different encodings: T1, T2 and T3. Encoding T1 and T2 are 16-bit long, and encoding T3 is 32-bit long.

Encoding T1 and T2 differs in the access to registers. In encoding T1, the two operand registers and the destination register can all be different. However, only the first 8 registers can be used. In contrast, encoding T2 supports access to all 16 of the registers, but the first operand and the destination register have to be the same.

Encoding T3 is more powerful than T1 and T2 that it exposes access to the barrel shifter, and supports full access to registers. It makes bit-shifting on-the-fly possible, and it is commonly used in addressing with scaling. It also allows the expression, such as addition with multiples of an operand, in one instruction.

To evaluate whether the encoding of an instruction affects the power consumption, the following kernel bodies are used. In these kernels, the same instruction `ADD r1, r0` are expressed in encoding T1, T2 and T3 respectively.

```
@ add_reg_T1
.hword 0b0001100000001001
@ add_reg_T2
.hword 0b0100010000000001
@ add_reg_T3_lsl0
.hword 0xeb01
.hword 0x0100
```

As shown in Fig. 4, three different encodings lead to statistically signifiant difference in terms of power consumption.

3.4 The Encodings of Subtraction Instructions

There are two different encodings for `SUB` (register), The first encoding T1 is similar to encoding T1 of `ADD` (register), and the second encoding T2 is similar to encoding T3 of `ADD` (register). In addition, there is an instruction `RSB`, which supports reverse subtraction with barrel shift. Intuitively, `SUB r1, r1, r0` is equivalent to `RSB r1, r0, r1`, but the difference between them in power consumption needs to be evaluated.

Kernel	Sample size	Mean	Standard deviation
`add_reg_T1`	20	1414594.3	701.9635
`add_reg_T2`	20	1411220.85	899.2575
`add_reg_T3_lsl0`	20	1402551.8	585.7135

(a) Power consumption of kernels

	`add_reg_T1`	`add_reg_T2`	`add_reg_T3_lsl0`
`add_reg_T1`	1.00E+00	1.90E-15	1.55E-38
`add_reg_T2`	1.90E-15	1.00E+00	1.28E-30
`add_reg_T3_lsl0`	1.55E-38	1.28E-30	1.00E+00

(b) P-values of kernels

Figure 4: Different encodings of the same addition operation

The techniques discussed in Section 3.3 can also be applied here, and the following kernel bodies are used.

```
@ sub_reg_T1
.hword 0b0001101000001001
@ sub_reg_T2
.hword 0b1110101110100001
.hword 0b0000000100000000
@ rsb_reg_T1
.hword 0b1110101111000000
.hword 0b0000000100000001
```

As shown in Fig. 5, the same subtraction calculation can be expressed in three different ways, and lead to statically significant difference in energy consumption.

Kernel	Sample size	Mean	Standard deviation
`sub_reg_T1`	20	1473407.9	1042.7334
`sub_reg_T2`	20	1458067.6	917.7737
`rsb_reg_T1`	20	1470169.55	784.7996

(a) Power consumption of kernels

	`sub_reg_T1`	`sub_reg_T2`	`rsb_reg_T1`
`sub_reg_T1`	1.00E+00	1.15E-35	3.69E-13
`sub_reg_T2`	1.15E-35	1.00E+00	4.31E-34
`rsb_reg_T1`	3.69E-13	4.31E-34	1.00E+00

(b) P-values of kernels

Figure 5: Different encodings of the same subtraction operation

3.5 Memory Operations

This approach can also be applied to memory operations. As a proof of concept, the `LDR` (register) and `STR` (register) instructions are evaluated. The following kernel bodies are used.

```
@ LDR r1, [r0, r2]
.hword 0b0101100010000001
@ STR r1, [r0, r2]
.hword 0b0101000010000001
```

As shown in Table 2, loading and storing to memory using register addressing have statistically significant difference in power consumption.

Kernel	Sample size	Mean	Standard deviation
`ldr_reg_T1`	20	1625625.45	875.6489
`str_reg_T1`	20	1664383.65	1021.7169
p-value	2.24E-51		

Table 2: Power consumption of memory operations

4 Conclusion

Being power efficient is an important aspect of computer systems design. Different factors involved in using instructions, such as choosing encodings, provide unique opportunities for conserving energy whiling archiving similar functionality.

In this report, I presented how power consumption measurements can be obtained on `32L476GDISCOVERY`. I showed the encodings of instructions can affect the energy consumption of the MCU.

4.1 Future Work

There are several potential extensions to this work that I would like to do in future.

4.1.1 Compiler Optimization Integration

System programming languages, such as C, are widely used to target applications to embedded devices. With the instruction-level energy consumption model, one could integrate this into compiler frameworks, such as GCC and LLVM. These compilers may provide compiler flags to optimise the generated code for minimum power consumption, small code size, etc.

4.1.2 External Power Measurement Units

In this work, the power measurement is done via the on-board ammeter, so that the whole measurement infrastructure is self-contained. However, this also implies that the code reading the measurement is running on the same MCU being measured. This might have negative impact on the reliability of the result we are getting.

One could connect an external ammeter to the pin 1 and 2 of jumper `JP5` of `32L476GDISCOVERY` to measure current directly. Alternatively, Cao et al.

[2012] used an external Hall effect sensor, and an Arduino board to get energy readings. This approach could be adapted to cross check with the readings from the on-board ammeter.

4.1.3 More Comprehensive Evaluation

In this work, I isolated individual instructions to reason about their power consumption characteristics. In future, my evaluation approach could be extended to real-world applications instead of synthetic homogeneous kernels. This could help us to find out whether the combination of instruction encodings makes difference in terms of energy consumption.

4.1.4 Establishing Power Consumption Models

This work only provides individual data points for each instruction encoding. The approach I used could be extended to establish power consumption models. These models can then be used to predict how much energy a program requires based the pattern of hardware usage, the size of function, frequency of branching, etc.

Acknowledgments

First, I would like to thank my supervisor, Ben Swift, for his patient guidance. I would also like to thank Ting Cao for pointing me to relevant resources. Her PhD thesis directly inspired this work. Thanks also to my friend, Brenda Wang, for helping me with the experiments and providing feedback for my writing.

References

Cao, T.; Blackburn, S. M.; Gao, T.; and McKinley, K. S., 2012. The yin and yang of power and performance for asymmetric hardware and managed software. In *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ISCA '12 (Portland, Oregon, 2012), 225–236. IEEE Computer Society, Washington, DC, USA. http://dl.acm.org/citation.cfm?id=2337159.2337185. (cited on page 8)