Evaluating Algorithms according to their Energy Consumption

Hannah Bayer and Markus Nebel

University of Kaiserslautern, Department of Computer Sciences,
Gottlieb-Daimler-Strasse, 67633 Kaiserslautern, Germany,
{h_bayer,nebel}@informatik.uni-kl.de

Abstract. This work deals with the evaluation of algorithms according to their energy consumption. So far it was a common belief that faster algorithms consume less energy than slower ones. This work presents results indicating that this is not universally valid. For this purpose an energy model shall be introduced which is used to determine the energy consumption of algorithms with regard to the input size. Thereafter the algorithms will be compared to each other regarding both to their run time and energy consumption.

Keywords: power consumption, analysis of algorithms, algorithm engineering

1 Motivation

Conventionally algorithm engineering is concerned with the run time and algorithms have therefore been evaluated with respect to their performance. Accordingly the run time of algorithms was the ultimate factor to be analyzed and optimized over the past years. However, over the years processors got faster and consumed more energy and the variety of fields where computers and embedded systems are used grew. So nowadays the power consumption of algorithms is an important factor to be taken into account.
There are many differently motivated reasons for trying to find ways to save energy consumed by computers which shall not be exhausted here. Existing solutions for minimizing energy consumption are multifaceted and span all components and architectural layers. There ACPI is to be mentioned, which allows the operating system to gain direct control over the power consumption as it can power down components after some time of inactivity. Another approach is to change the voltage according to the load of the system like in [Hsu03] and thus lower the power consumption. A lower consumption can naturally be realized through optimizations of hardware too but this seldom had been the main goal for the development of new processors. Finally the optimization of software shall be contemplated where previous works have already developed methods to lower the power consumption of algorithms.The main idea there is to optimize the process of compiling programs written in higher languages to assembler or machine

code. As an example [LKHcT00] should be mentioned were the consumption is lowered by choosing a special alignment of the instructions. Some of those works as [SKWM01,LTMF95,The05,SL01,CKL00,TMW94,TMWL96,CKL02,GN00] do not directly derive methods to reduce the power consumption but present techniques and models to calculate the actual power consumed by an algorithm. Additionally there are papers concerning tools ([HKS$^+$07] and [SC01]) simulating the energy consumption.
In the sequel it shall be described how the simulation tool XEEMU ([HKS$^+$07]) can be used to compare the power consumption of algorithms and a theoretical model shall be introduced which allows for the quantification of power consumed during the execution of algorithms written in assembler code. Even if it is common belief that faster algorithms need less energy – and knowledge of their run time therefore is sufficient – the simulations and calculations based on the model to be introduced will be used to analyze the expected energy consumption of several algorithms from searching and sorting. As a surprise the results show that there are pairs of algorithms where the faster one consumes more energy.
The rest of this paper is organized as follows. First some informations about the general approach will be given, in section 3 the preconditions for the simulation will be elucidated and the results of the simulations performed will be presented. Section 4 contains the development of the theoretical model whereas section 5 engages in the discussion of the results of this model.

2 Preliminaries

The algorithms to be analyzed are taken out of the range of searching an sorting algorithms. To refer to a consistent base the assembler code MIX [Knu98] will be used to describe the different algorithms for the theoretical model as well as for the simulation. As MIX has some instructions that do not exist the modern processors it is required that those instructions are "simulated" by two ore more independent instructions on the processors to be simulated and those to be modelled . To retrieve concrete data two processors will be used, the proprietary DSP from Fujitsu and the ARM7TDMI commonly used in high end embedded systems whereas the simulation tool XEEMU can be used to simulate the power consumption of an ARM5-processor. Even if the power consumption of processors for embedded systems is already optimized somewhat comparing to other processores nevertheless it will be interesting, for the design of embedded systems did potentially take other possibilities of reduction of power consumption into account and maybe the prospects of those have already been exhausted.

3 Simulation and first results

To gain a first impression of the power consumption of algorithms we will present the results from the simulation of certain searching algorithms originated from simulation with XEEMU ([HKS$^+$07]) which were derived during the work of a student [Bra08]. For every algorithm the average run time and the average power consumption were derived from several passes of simulations. As the effects of cache misses change with different cache hierachies and different sizes of cache the effects do not merely depend on the algorithm itself. Hence, the cache size in XEEMU is set thus that no cache misses will take place after an initial process of once reading all elements (the power consumption of this process will be subtracted from the total power consumption). For the simulation of the average power consumption two versions were derived; one including and one excluding the power consumption of the cache. The latter will be interesting as the cache size is (as described above) chosen to be large and as a larger cache consumes more power (for example for cache management) the total power consumption is thus fairly dominated by the power consumption of the cache. In the sequel the average case results for three different types of sequential sorting algorithms will be presented as well as results for *Binary Search*, *Uniform Binary Search* and *Fibonaccian Search*. In Figure 1 it can be seen that for the run time the *Sequential Search* is significantly slower than the other two algorithms and it shows that the power consumption of all three algorithms appear to be the same if the consumption of the cache is taken into account. If this consumption is substracted it shows that there are differences between all three algorithms and thus shows that even if the *Quicker Sequential Search* is not significantly faster than the *Quick Sequential Search* it consumes less power. The Figure 2 pictures

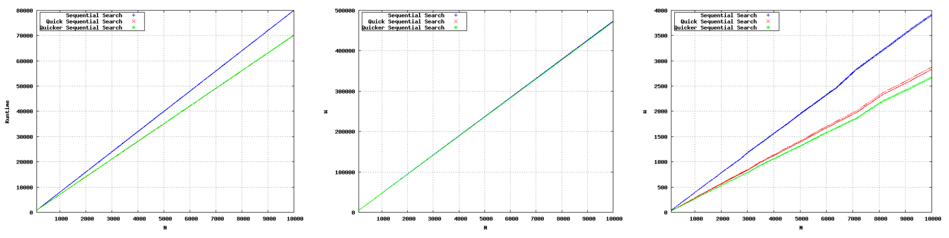

Fig. 1: Average Run time /Average Power consumption with cache consumption /without cache consumption (Appendix Figures 7,8 and 9).

the run time and the power consumption of *Binary Search*, *Uniform Binary Search* and *Fibonaccian Search*. Relative to the run time the *Fibonaccian Search* is clearly the worst of the three algorithms whereas it cannot be stated clearly which one of the other two algorithms is the faster one. Considering the total power consumption including the consumption of the cache Figure 2 shows that the *Fibonaccian Search* is better than the *Binary Search* which should be chosen over the *Uniform Binary Search*. If one does not take the consumption of the cache into account the *Fibonaccian Search* seems not to be the best but not the worst either whereas it shows that the *Binary Search* seems to consume more power than the *Uniform Binary Search* at most input sizes.
Thus it can be seen that the correlation between run time an power consumption

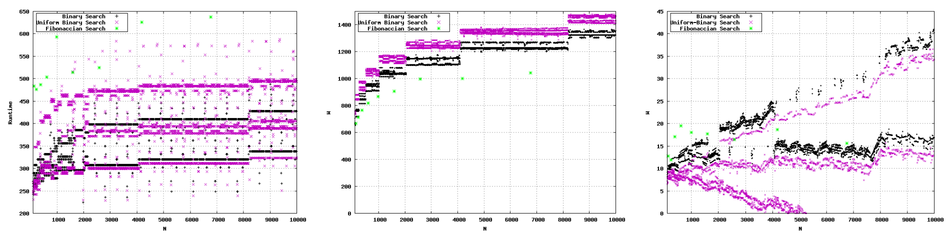

Fig. 2: Average Run time /Average Power consumption with cache consumption /without cache consumption (Appendix Figures 10,11 and 12).

is not necessarily existing. As the simulation of algorithms does take a lot of time especially for the average case further examinations were made with the help of a theoretical model that can be directly applied to algorithms written in assembler code.

4 Energy Model

We will continue by describing our model and its derivation. In [LTMF95] a model based on a risc-architecture is presented. According to this work, the main power consumption can be divided into two parts. One part is based on the instructions and the other one is based on the actual data used. Different instructions are performed by different components of the processor like the ALU or the multiplier or by a combination of components. As those components consume different amounts of power this results in different power consumptions for the variety of instructions. Furthermore, since not every component of the processor is used for every instruction components can be switched off when they are not used. Switching components off and on consumes energy and is therefore required to be noticed in an energy model. The power consumption in the CPU resulting from the processed data depends on the number of ones in the binary representation in the actual input and on the hamming-distances between two following sets of data the so-called bit-toggling.
Another model presented in [SKWM01] basically discriminates between two different kinds of power consumption, the consumption of the instruction itself and the consumption of the overhead resulting of the on and off switching of components. Furthermore this model does account for the consumption of pipeline stalls and cache misses.
The model shall analyze the power consumption as independently as possible from the data processed, therefore all mere data-dependend power consumption shall be left unregarded. Furthermore Pipeline stalls will not be accounted for as it is hard to calculate how often they are going to occur on average for a given algorithm with a specific size of input and as the existing knowledge regarding the power consumption of pipeline stalls is limited. In contrast the

existing informations about power consumption referring to cache misses could be used easily, but average case analysis of cache misses is yet not very common and therefore the existing results concerning their occurence are few and not very exact. Hence, only the consumption of the instructions and the overhead between two instructions will be taken into account.The conclusions drawn later all are subject to the presumption that pipeline stalls and cache misses do not alter the results if the effects of those would be considered in the model.
Further on it is presumed that the instructions of the processor to be modeled can be ordered into groups where all instruction contained in one group do have a very similar power consumption and the overhead between an instruction and another is nearly the same for all other instructions of the same group.
As the algorithms to be analyzed are written in the assembly language MIX, every instruction of this language will be filed into one such group. Since the goal is an energy model which can be applied to a wide variety of processors those groups are defined in a way that not only the instruction set of one processor will fit into it. This results in some groups having the same power consumption for a specific processor and the same overhead for the switching from and to other groups. The actual grouping is listed in the appendix.
We will use the following notation (as denoted in the example below): Every group is assigned with a factor standing for the value of the power consumption, for the overhead between two groups the factor will be noted as the names of the factors of the two groups separated by a colon. The value of the factors change according to the processor to be modeled; the actual values for the two processors are listed in the appendix as well. Below the application of the energy model to *Sequential Search* will be discussed to exemplify how algorithms were analyzed. The algorithm in Figure 3 is directly adopted from [Knu98] supplemented with the overhead between two instruction and the group-factor of instructions and overheads.

line	label	instruction	number of executions	group-factor
1	START	LDA K	1	α_1
2			1	$\alpha_1 : \alpha_6$
3		ENT1 1-N	1	α_6
4			1	$\alpha_6 : \alpha_8$
5	2H	CMPA KEY+N,1	C	α_8
6			C	$\alpha_8 : \alpha_{11}$
7		JE SUCCESS	C	α_{11}
8			C-S	$\alpha_{11} : \alpha_7$
9		INC1 1	C-S	α_7
10			C-S	$\alpha_7 : \alpha_{11}$
11		J1NP 2B	C-S	α_{11}
12	FAILURE	EQU *	1-S	

Overhead resulting of Jumps: line 11 to line 5: C-1 times $\alpha_{11} : \alpha_8$

Fig. 3: MIX Code amended for the analysis.

Initially the accurate values for the occurrences of the instructions and the overheads are calculated based on input size N. In addition the run time is evaluated.

Analysis for a successfull search

The following formulas are adopted from [Knu98].

$S = 1$

The number of comparisons is based on the assumption that every element of the searched set of elements is searched with the same probability.

$C = \frac{N+1}{2}$

Run time in u (Units of time) [Knu98]: $(2,5 \cdot N + 3,5)\ u$

Average Power consumption

To analyze the power consumption of an algorithm one does only have to add the quantities of the factors which depend on the size of input. E = $\alpha_1 + \alpha_1 : \alpha_6 + \alpha_6 + \alpha_6 : \alpha_8 + C \cdot (\alpha_8 + \alpha_8 : \alpha_{11} + \alpha_{11}) + (C - S)(\alpha_{11} : \alpha_7 + \alpha_7 + \alpha_7 : \alpha_{11} + \alpha_{11}) + (C - 1)(\alpha_{11} : \alpha_8)$

With the values for the group-factors from the appendix the following power consumption of the two processors can be derived:

DSP $E = (92,55 + 109,75 \cdot N)mA$

ARM7TDMI $E = (113.025 + 114,165 \cdot N)mA$

Now it is easy to compare algorithms regarding their power consumption. To visualize this the run time and the power consumption of the algorithms to be compared have been put into different plots (see for example Figure 4), where the x-axis is the input size and the y-axis is the run time or respectively the power consumption. It can be seen easily which algorithm is faster and which algorithm has the lower power consumption.
Another interesting fact to consider is the leakage power which is the power that is always consumed whether the processor is idle or not. The crucial point is now to learn if an idle processor would be turned off immediately the leakage power would affect the above statements concerning the evaluation according to the power consumption. If there are two algorithms (with the slower one consuming less energy) the first with the run time t and the power consumption e and the second respectively with t' and e' one only needs to solve the equation

$$e + t \cdot l = e' + t' \cdot l$$

for l. Thus with l one gets the leakage power it would take to delete the observed discrepancy between run time and energy consumption. If l is larger than the leakage power of the specific processor the discrepancy holds if leakage power is taken into account.

5 Results

For most problems it does apply that the fastest algorithm is the one with the lowest power consumption like for example for *quicksort* compared with *merge*

sort or *heapsort*. But there are exceptions that cannot be disregarded. Those exceptions can be divided into different types. First are those algorithms that are faster but consume more energy than other algorithms for certain scopes of input sizes. Secondly those that are faster but consume more energy for all input sizes and last there are those where the statement of the second type holds true even when leakage power is taken into account. Good examples of the first kind are given by the algorithms for sequential search.
The Figure 4 illustrates the average case run time and the power consumption of *Sequential Search*, *Quick Sequential Search* and *Quicker Sequential Search* as described in [Knu98]. Whereas one cannot see a significant difference between the run time and the power consumption for great input sizes this is different for small input sizes as shall be seen in the sequel.

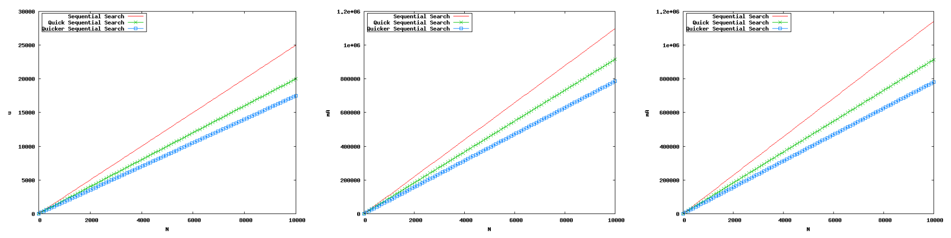

Fig. 4: Average Run time/ Average Power consumption DSP/ Average Power consumption ARM7TDMI (Appendix Figures 13,14 and 15).

The following table depicts the intersection points of the graphs regarding the size of input for the three types of sequential searches and thus shows the scope of input sizes, where the faster algorithm consumes more energy than the slower one.

	Run time	Power consumpt. DSP	Power consumpt. ARM7TDMI
Sequential Search & Quick Sequ. S.	9	6,58	7,057
Sequential Search & Quicker Sequ. S.	6,667	7,077	7,936
Quick Sequ. Search & Quicker Sequ. S.	2	7,763	9,443

As shown in the table above the intersection points of the graphs of the run time and the power consumption do not result from the same input sizes. For example whereas *"Quicker Sequential Search"* is to be preferred to *"Quick Sequential Search"* at almost any input size regarding the run time the same does not hold for the energy consumption. Therefore it would be better to use *"Quick Sequential Search"* for smaller input values and to switch to *"Quicker Sequential Search"* for larger input values to save energy.

The algorithms *"Uniform Binary Search"* and *"Fibonaccian Search"* shall exemplify the second kind. In Figure 5 one can see that with regard to the run time

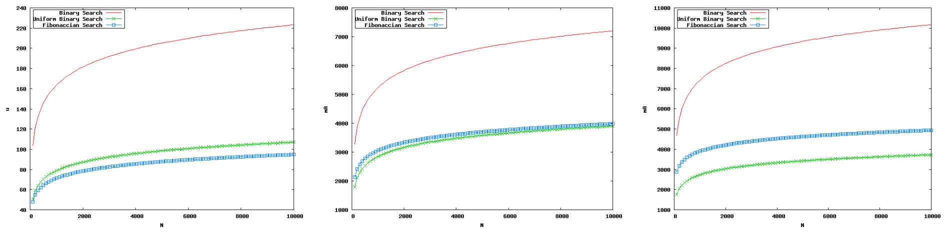

Fig. 5: Average Run time/ Average Power consumption DSP/ Average Power consumption ARM7TDMI (Appendix Figures 16,17 and 18).

the *"Fibonaccian Search"* should be preferred to the *"Uniform Binary Search"* this cannot be said regarding the power consumption. Looking at the consumption of the DSP the *"Fibonaccian Search"* is slightly worse but with regard to the ARM7TDMI the discrepancy becomes significant. Furthermore it is to be mentioned that the graphs plotted above are only those for the average case but the basic message is the same for the worst case.
If leakage power is accounted for the result changes for the power consumption of the DSP, the *"Fibonaccian Search"* does no longer consume more energy then the *"Uniform binary search"*. Contrary to that the result does not change for the ARM7TDMI.
Similarly significant is the comparison of *"Straight Insertion Sort"*, *"Straight Selection Sort"* and *"List Insertion Sort"*.

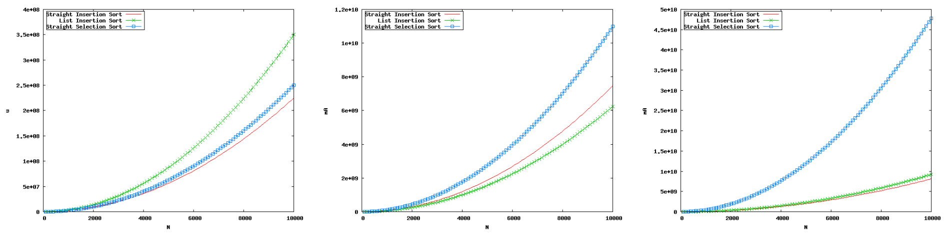

Fig. 6: Average Run time/ Average Power consumption DSP/ Average Power consumption ARM7TDMI (Appendix Figures 19,20 and 21).

In Figure 6 again the average case is plotted. Similar to the illustration above for the binary searches it can be stated for the sorting algorithms illustrated that

the fastest algorithms is not the one with the lowest energy consumption. Regarding the run time "List Insertion Sort" is the worst algorithm but regarding the power consumption on the DSP it is the best and "Straight Selection Sort" is the worst regarding the power consumption of both DSP and ARM7TDMI but has definitely not the longest run time. Again the shown plots pictures the average case but in difference to the binary search algorithms the average case is unlike the worst case but nonetheless in the worst case the order of the algorithms changes as well from run time to energy consumption.
The results for the quadratic sorting algorithms do not change by taking the leakage power into account and thus the algorithms are of the third kind mentioned above.

6 Resume

Our analysis and simulation has proven that, assuming our model to be realistic, the faster algorithm is not necessarily the one with the lower power consumption. Even if one takes account of leakage power the gained results almost always keep the same. Considering using different algorithms to save energy requires to analyze the relevant algorithms according to the specific processor. For some problems an appropriate use of the right algorithm could save a great amount of energy especially if a particular problem is solved very frequently.
As the obtained results are based on a theoretical model and the simulation of few algorithms on one processor the next step will be to affirm the results on basis of more simulation. Further on there are some options to improve the existing model and data. To examine the behaviour of algorithms to the energy consumption on specific processors more closely it would be necessary to explore the effect of cache misses and pipeline stalls. Another interesting option would be to apply the existing energy model to more processors especially to processors not designed for embedded systems.

References

[Bra08] Tobias Braun. Energieverbrauch von Suchalgorithmen. Projektarbeit, Technische Universität Kaiserslautern, September 2008.

[CKL00] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy consumption measurement and analysis: case study of ARM7TDMI. In *ISLPED '00: Proceedings of the 2000 international symposium on Low power electronics and design*, pages 185–190, New York, NY, USA, 2000. ACM.

[CKL02] Naehyuck Chang, Kwanho Kim, and Hyung Gyu Lee. Cycle-accurate energy measurement and characterization with a case study of the ARM7TDMI. *IEEE Trans. Very Large Scale Integr. Syst.*, 10(2):146–154, 2002.

[GN00] S. Gupta and F. Najm. Power Modeling for High-level Power Estimation. In *1EEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 8, pages 18–29, 2000.

[HKS$^+$07] Zoltán Herczeg, Ákos Kiss, Daniel Schmidt, Norbert Wehn, and Tibor Gyimóthy. XEEMU: An Improved XScale Power Simulator. In *PATMOS*, pages 300–309, 2007.

[Hsu03] Chung-Hsing Hsu. *Compiler-directed dynamic voltage and frequency scaling for cpu power and energy reduction.* PhD thesis, New Brunswick, NJ, USA, 2003.

[Knu98] Donald E. Knuth. *The Art of Computer Programming*, volume 3 Sorting and Searching. Addison Wesley, 2. edition, 1998.

[LKHcT00] Chingren Lee, Jenq Kuen, Lee Tingting Hwang, and Shi chun Tsai. Compiler optimization on instruction scheduling for low power. In *In 13th International Symposium on System Synthesis. ACM, Septermber*, pages 55–60. ACM Press, 2000.

[LTMF95] Mike Tien-Chien Lee, Vivek Tiwari, Sharad Malik, and Masahiro Fujita. Power analysis and low-power scheduling techniques for embedded DSP software. In *ISSS '95: Proceedings of the 8th international symposium on System synthesis*, pages 110–115, New York, NY, USA, 1995. ACM.

[SC01] Amit Sinha and Anantha P. Chandrakasan. JouleTrack: a web based tool for software energy profiling. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 220–225, New York, NY, USA, 2001. ACM.

[SKWM01] Stefan Steinke, Markus Knauer, Lars Wehmeyer, and Peter Marwedel. An accurate and fine grain instruction-level energy model supporting software optimizations. In *in Proc. Int. Wkshp Power and Timing Modeling, Optimization and Simulation (PATMOS)*, 2001.

[SL01] Sang Lyul Min Sheayun Lee, Andreas Ermedahl. An Accurate Instruction-level Energy Consumption Model for Embedded Risc Processors. *ACM SIGPLAN Notices*, 36, August 2001.

[The05] Michael Theokaridis. Measuring Energy consumption of ARM7TDMI Processor Instructions. Master's thesis, Technische Universität Dortmund, Juni 2005.

[TMW94] V. Tiwari, S. Malik, and A. Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, 1994.

[TMWL96] Vivek Tiwari, Sharad Malik, Andrew Wolfe, and Mike Tien-Chien Lee. Instruction level power analysis and optimization of software. *J. VLSI Signal Process. Syst.*, 13(2-3):223–238, 1996.

7 Appendix

A Larger Pictures

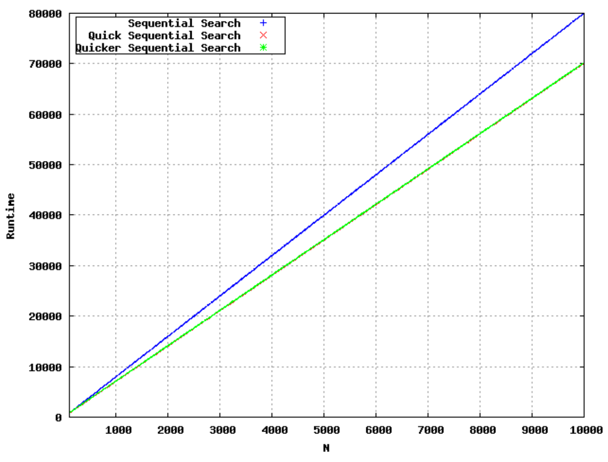

Fig. 7: Average Run time (XEEMU): Sequential Search, Quick Sequential Search, Quicker Sequential Search.

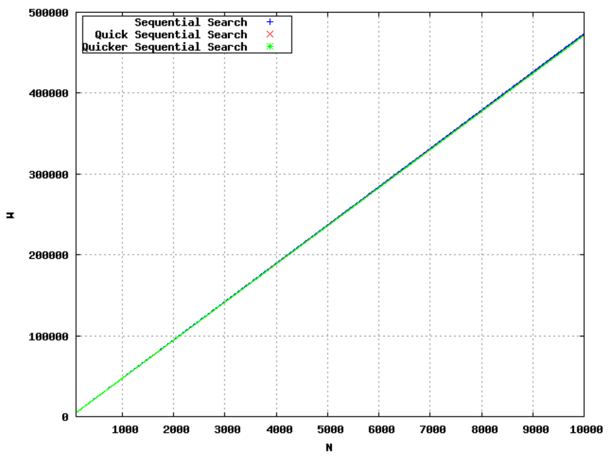

Fig. 8: Average Power Consumption with cache (XEEMU): Sequential Search, Quick Sequential Search, Quicker Sequential Search.

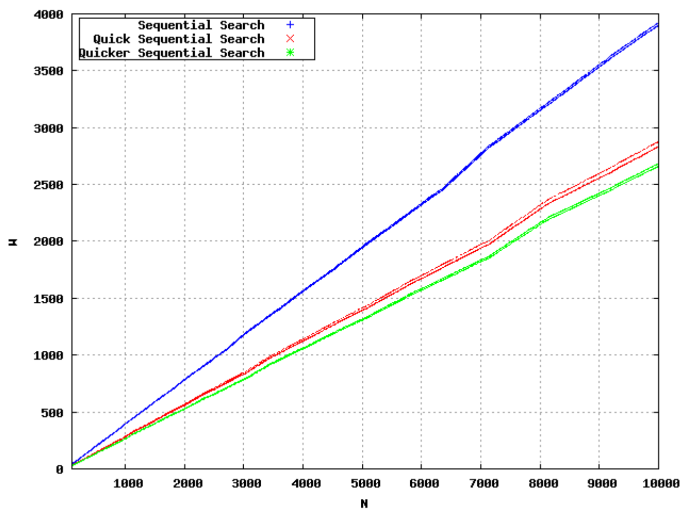

Fig. 9: Average Power Consumption without cache (XEEMU): Sequential Search, Quick Sequential Search, Quicker Sequential Search.

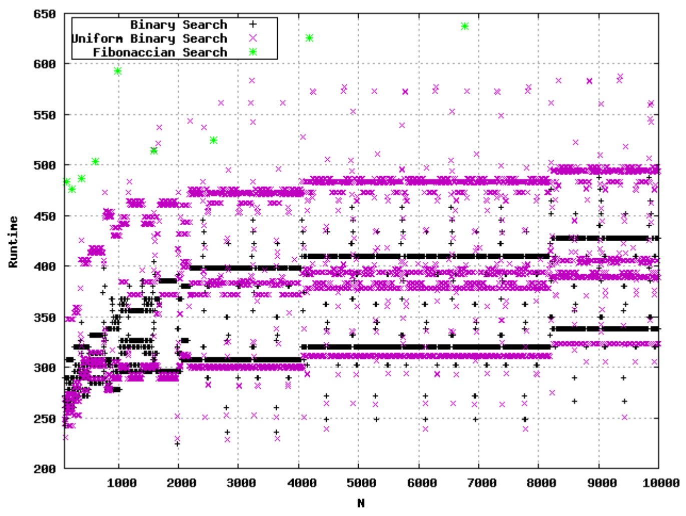

Fig. 10: Average Run time (XEEMU): Binary Search, Uniform Binary, Search Fibonaccian Search.

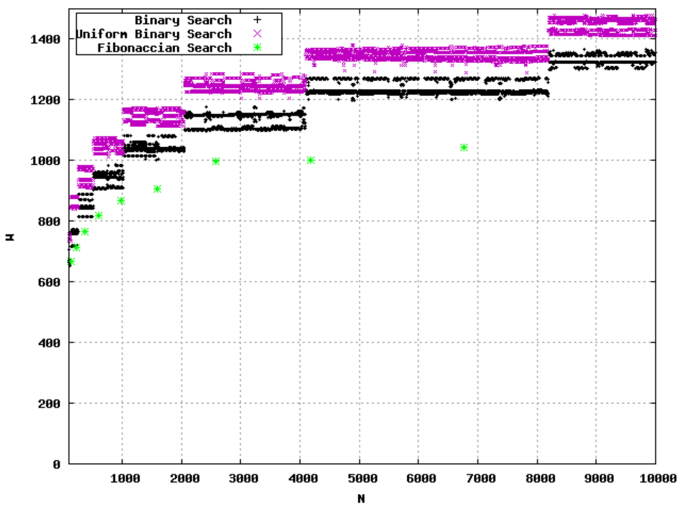

Fig. 11: Average Power Consumption with cache (XEEMU): Binary Search, Uniform Binary, Search Fibonaccian Search.

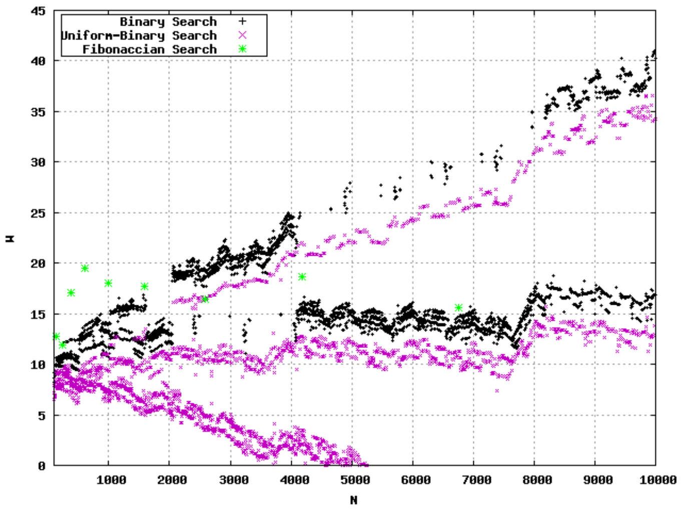

Fig. 12: Average Power Consumption without cache (XEEMU): Binary Search, Uniform Binary, Search Fibonaccian Search.

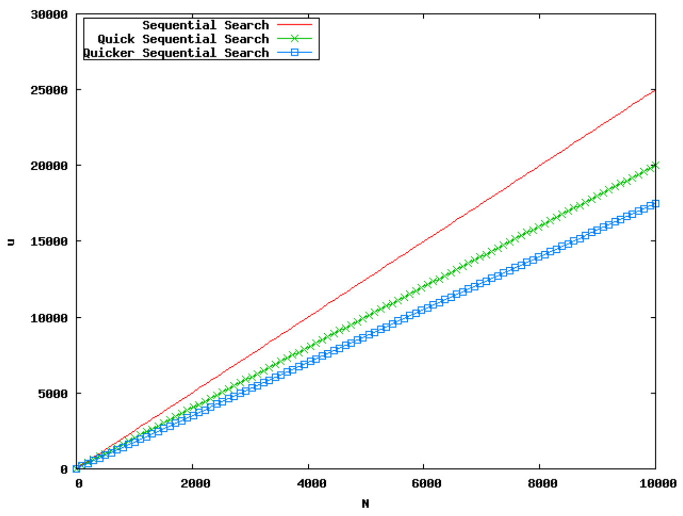

Fig. 13: Average Run time (MIX): Sequential Search, Quick Sequential Search, Quicker Sequential Search

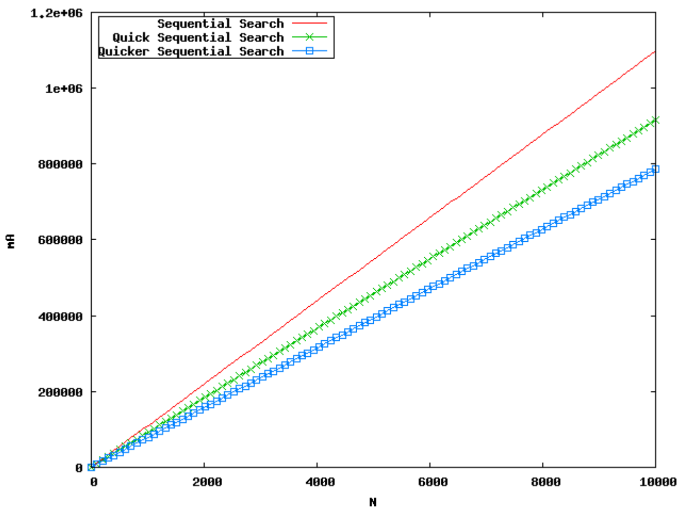

Fig. 14: Average Power Consumption DSP: Sequential Search, Quick Sequential Search, Quicker Sequential Search

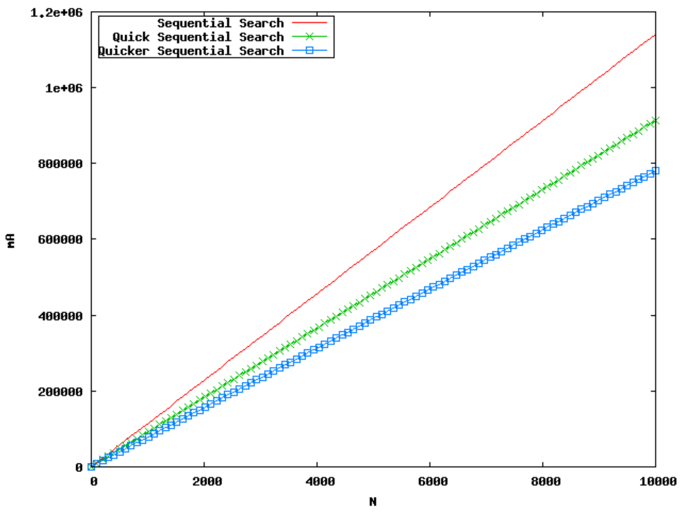

Fig. 15: Average Average Power Consumption ARM7TDMI: Sequential Search, Quick Sequential Search, Quicker Sequential Search

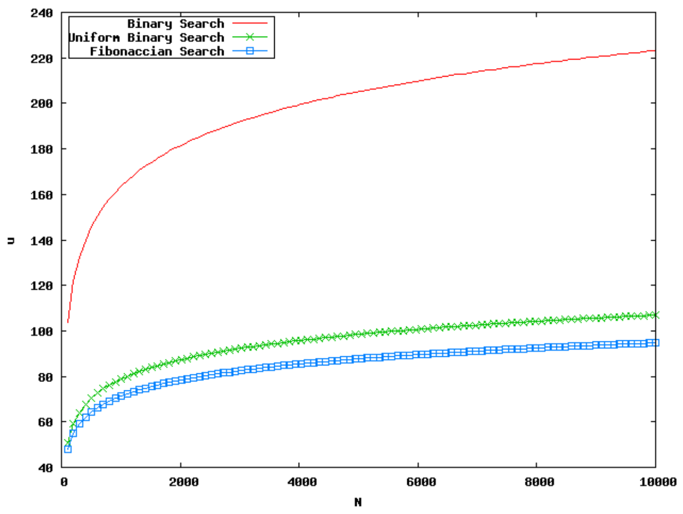

Fig. 16: Average Run time (MIX): Binary Search, Uniform Binary Search, Fibonaccian Search

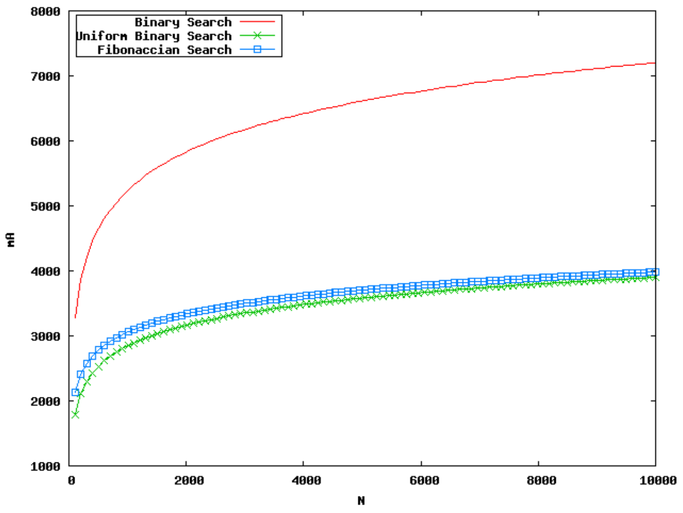

Fig. 17: Average Power Consumption DSP: Binary Search, Uniform Binary Search, Fibonaccian Search

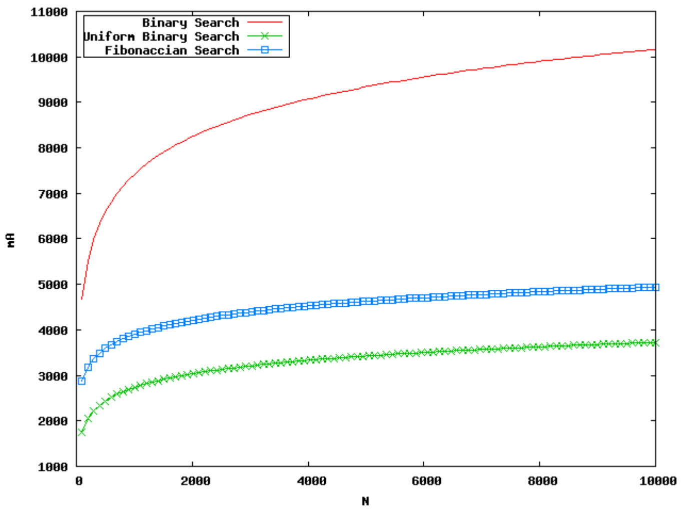

Fig. 18: Average Power Consumption ARM7TDMI: Binary Search, Uniform Binary Search, Fibonaccian Search

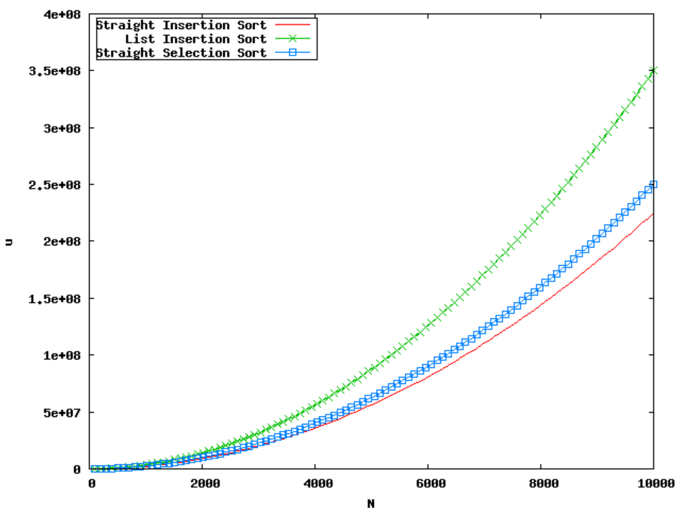

Fig. 19: Average Run time (Mix): Straight Insertion Sort, List Insertion Sort, Straight Selection Sort.

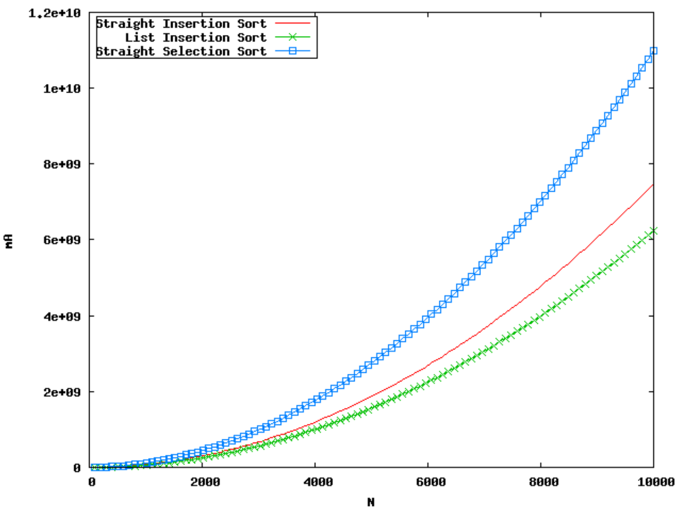

Fig. 20: Average Power Consumption DSP: Straight Insertion Sort, List Insertion Sort, Straight Selection Sort.

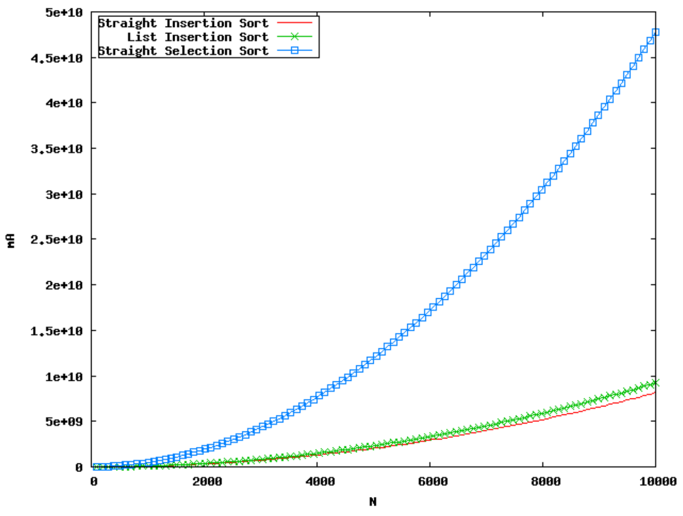

Fig. 21: Average Power Consumption ARM7TDMI: Straight Insertion Sort, List Insertion Sort, Straight Selection Sort.

B Division of instructions

Following the instructions of the MIX-languages are divided into functional groups.

1. Load-instructions for registers A and X weighted with factor α_1:
   LDA
   LDX
   LDAN
   LDXN

2. Load-instructions for other registers and ans save-instructions for all register weighted with factor α_2:
   LDi
   LDiN
   STA
   STX
   STi
   STJ
   STZ

3. Add- and subtract-instruction weighted with factor α_3:
   ADD
   SUB

4. Multiply-instructions weighted with factor α_4:
   MUL

5. Divide-instructions weighted with factor α_5:
   DIV

6. Load-instructions for Immediates to a register weighted with factor α_6:
   ENTA
   ENTX
   ENTi
   ENNA
   ENNX
   ENNi

7. Addition/subtraction of Immediates weighted with factor α_7:
   INCA
   INCX
   INCi
   DECA
   DECX
   DECi

8. Comparisons weighted with factor α_8:
   CMPA
   CMPX
   CMPi

9. Unconditoned jump weighted with factor α_9:
   JMP

10. Unconditioned jump with saving of the jump-address weighted with factor α_{10}
    JSJ

11. Conditioned jump weighted with factor α_{11}:
    JOV
    JNOV
    JL
    JE
    JG
    JGE
    JNE
    JLE
    JAN
    JAZ
    JAP
    JANN
    JANZ
    JANP
    JXN
    JXZ
    JXP
    JXNN
    JXNZ
    JXNP
    JiN
    JiZ
    JiP
    JiNN
    JiNZ
    JiNP

12. Shift-operation on one register weighted with factor α_{12}:
    SLA
    SRA

13. Shift-operation on two registers weighted with factor α_{13}:
    SLAX

SRAX
SLC
SRC

14. Move-operation within the central memory weighted with factor α_{14}:
MOVE

15. NOP weighted with 0.

16. HLT weighted with 0.

17. I-/O-Operationen weighted with factor α_{15}:
IN
OUT
IOC

18. Jump-operations testing the secondary memory weighted with factor α_{16}:
JRED
JBUS

19. Conversional operations weighted with factor α_{17}:
NUM
CHAR

20. Moving of register contents weighted with factor α_{18}:
(i,j Registernummer)
ENTA 0,j
ENTX 0,j
ENTi 0,j
ENNA 0,j
ENNX 0,j
ENNi 0,j

21. Moving of register contents + adding of an immediate weighted with factor α_{19}:
(i,j Registernummer, $l \neq 0$)
ENTA l,j
ENTX l,j
ENTi l,j
ENNA l,j
ENNX l,j
ENNi l,j

C Concrete Values for the DSP

The values for the DSP instructions are extracted from [SKWM01]. There the instructions were already ordered into groups named LAB, MOV1, MOV2, ASL and LDI. Below the factors of the instruction groups of the MIX-language are assigned to appropriate values.

C.1 Instructions

$\alpha_1 = LAB = 36,5\ mA$
$\alpha_2 = MOV2 = 18,4\ mA$
$\alpha_3 = LAB + LAB : ASL + ASL = 73,9\ mA$
$\alpha_4 = LAB + LAB : MAC + MAC = 68,7\ mA$
$\alpha_5 = LAB + LAB : ASL + ASL = 73,9\ mA$
$\alpha_6 = LDI = 19,4\ mA$
$\alpha_7 = ASL = 16,5\ mA$
$\alpha_8 = LAB + LAB : ASL + ASL = 73,9\ mA$
$\alpha_9 = MOV2 = 18,4\ mA$
$\alpha_{10} = MOV2 = 18,4\ mA$
$\alpha_{11} = MOV2 = 18,4\ mA$
$\alpha_{12} = ASL = 16,5\ mA$
$\alpha_{13} = 2 \cdot ASL + ASL : ASL = 36,6\ mA$
$\alpha_{14} = 2 \cdot MOV2 + MOV2 : MOV2 = 62,4\ mA$
α_{15} was not used for this paper.
$\alpha_{16} = 2 \cdot MOV2 + MOV2 : MOV2 = 62,4\ mA$
$\alpha_{17} = ASL = 16,5\ mA$
$\alpha_{18} = MOV1 = 19,8\ mA$
$\alpha_{19} = MOV1 + MOV1 : ASL + ASL = 46,8\ mA$

C.2 Overhead

The following table presents the values for the overhead between two instructions. It should be noted that as one MIX-instruction can consist of several DSP-instruction the order of the instructions is relevant.

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}	α_{11}	α_{12}	α_{13}	α_{14}	α_{15}	α_{16}	α_{17}	α_{18}	α_{19}
α_1	2, 5	12, 2	2, 5	2, 5	2, 5	13, 7	20, 9	2, 5	12, 2	12, 2	12, 2	20, 9	20, 9	12, 2	?	12, 2	20, 9	1, 9	1, 9
α_2	12, 2	25, 6	12, 2	12, 2	12, 2	6, 3	26, 7	12, 2	25, 6	25, 6	25, 6	26, 7	26, 7	25, 6	?	25, 6	26, 7	18, 3	18, 3
α_3	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_4	15	22, 2	15	15	15	6, 0	8, 0	15	22, 2	22, 2	22, 2	8, 0	8, 0	22, 2	?	22.2	8, 0	3, 8	3, 8
α_5	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_6	13, 7	6, 3	13, 7	13, 7	13, 7	3, 6	10, 8	13, 7	6, 3	6, 3	6, 3	10, 8	10, 8	6, 3	?	6, 3	10, 8	15, 5	15, 5
α_7	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_8	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_9	12, 2	25, 6	12, 2	12, 2	12, 2	6, 3	26, 7	12, 2	25, 6	25, 6	25, 6	26, 7	26, 7	25, 6	?	25, 6	26, 7	18, 3	18, 3
α_{10}	12, 2	25, 6	12, 2	12, 2	12, 2	6, 3	12, 2	12, 2	25, 6	25, 6	25, 6	26, 7	26, 7	25, 6	?	25, 6	26, 7	18, 3	18, 3
α_{11}	12, 2	25, 6	12, 2	12, 2	12, 2	6, 3	26, 7	12, 2	25, 6	25, 6	25, 6	26, 7	26, 7	25, 6	?	25, 6	26, 7	18, 3	18, 3
α_{12}	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_{13}	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_{14}	12, 2	25, 6	12, 2	12, 2	12, 2	6, 3	26, 7	12, 2	25, 6	25, 6	25, 6	26, 7	26, 7	25, 6	?	25, 6	26, 7	18, 3	18, 3
α15																			
α_{16}	12, 2	25, 6	12, 2	12, 2	12, 2	6, 3	26, 7	12, 2	25, 6	25, 6	25, 6	26, 7	26, 7	25, 6	?	25, 6	26, 7	18, 3	18, 3
α_{17}	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26, 7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5
α_{18}	1, 9	18, 3	1, 9	1, 9	1, 9	15, 5	0, 5	1, 9	18, 3	18, 3	18, 3	10, 5	10, 5	18, 3	?	18, 3	10, 5	4, 0	4, 0
α_{19}	20, 9	26, 7	20, 9	20, 9	20, 9	10, 8	3, 6	20, 9	26, 7	26, 7	26.7	3, 6	3, 6	26, 7	?	26, 7	3, 6	10, 5	10, 5

D Concrete values for the ARM7TDMI

The values for the ARM7TDMI are derived from [The05]. There all instructions were listed and therefor the values following are generated from the average value for the several groups.

D.1 Instructions

$\alpha_1 = 46,5\ mA$
$\alpha_2 = 50,18\ mA$
$\alpha_3 = 46,5 + 2,05 + 41,35 = 89,9\ mA$
$\alpha_4 = 46,5 + 2,5 + 50,15 = 99,15\ mA$
α_5 was not used for this paper.
$\alpha_6 = 41,3\ mA$
$\alpha_7 = 41,6\ mA$
$\alpha_8 = 41,2 + 2,05 + 46,5 = 89,75\ mA$
$\alpha_9 = 42,9\ mA$
$\alpha_{10} = 42,3\ mA$
$\alpha_{11} = 41,69\ mA$
$\alpha_{12} = 43,7\ mA$
$\alpha_{13} = 45,03\ mA$
$\alpha_{14} = 100,37 + 0,4 = 100,77\ mA$
α_{15} was not used for this paper.
$\alpha_{16} = \alpha_1 + \alpha_1 : \alpha_{11} + \alpha_{11} = 46,5 + 2,05 + 41,69 = 90,24\ mA$
α_{17} was not used for this paper.
$\alpha_{18} = \alpha_6 + \alpha_6 : \alpha_3 + \alpha_3 = 41,3 + 3,8 + 89,9 = 135\ mA$
$\alpha_{19} = \alpha_6 + \alpha_6 : \alpha_3 + \alpha_3 = 135\ mA$

D.2 Overhead

As above the order of the instructions can change the overhead.

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}	α_{11}	α_{12}	α_{13}	α_{14}	α_{15}	α_{16}	α_{17}	α_{18}	α_{19}
α_1	$0,4$	$0,4$	$0,4$	$0,4$	?	$2,05$	$2,05$	$0,4$	$2,05$	$2,05$	$2,05$	2	2	$0,4$	?	$0,4$	?	$2,05$	$2,05$
α_2	$0,4$	$0,4$	$0,4$	$0,4$	?	$2,05$	$2,05$	$0,4$	$2,05$	$2,05$	$2,05$	2	2	$0,4$	?	$0,4$	?	$2,05$	$2,05$
α_3	$2,05$	$2,05$	$2,05$	$2,05$	?	$3,85$	$0,2$	$2,05$	$3,85$	$3,85$	$3,85$	$3,3$	$3,3$	$2,05$	?	$2,05$	?	$2,05$	$2,05$
α_4	$2,05$	$2,05$	$2,05$	$2,05$	?	$3,85$	$0,2$	$2,05$	$3,85$	$3,85$	$3,85$	$3,3$	$3,3$	$2,05$	?	$2,05$	?	$2,05$	$2,05$
α_5																			
α_6	$2,05$	$2,05$	$2,05$	$2,05$	?	$0,2$	$3,85$	$2,05$	$0,2$	$0,2$	$0,2$	$3,3$	$3,3$	$3,85$	?	$3,85$	?	$0,2$	$0,2$
α_7	$2,05$	$2,05$	$2,05$	$2,05$	?	$3,85$	$0,2$	$2,05$	$3,85$	$3,85$	$3,85$	$3,3$	$3,3$	$2,05$	?	$2,05$	?	$2,05$	$2,05$
α_8	$2,05$	$2,05$	$2,05$	$2,05$	?	$3,85$	$0,2$	$2,05$	$3,85$	$3,85$	$3,85$	$3,3$	$3,3$	$2,05$	?	$2,05$	?	$2,05$	$2,05$
α_9	$2,05$	$2,05$	$2,05$	$2,05$	?	$0,2$	$3,85$	$2,05$	$0,2$	$0,2$	$0,2$	$3,3$	$3,3$	$3,85$	?	$3,85$	?	$0,2$	$0,2$
α_{10}	$2,05$	$2,05$	$2,05$	$2,05$	?	$0,2$	$3,85$	$2,05$	$0,2$	$0,2$	$0,2$	$3,3$	$3,3$	$3,85$	?	$3,85$	?	$0,2$	$0,2$
α_{11}	$2,05$	$2,05$	$2,05$	$2,05$	?	$0,2$	$3,85$	$2,05$	$0,2$	$0,2$	$0,2$	$3,3$	$3,3$	$3,85$	?	$3,85$	?	$0,2$	$0,2$
α_{12}	2	2	2	2	?	$3,3$	$3,3$	2	$0,2$	$0,2$	$0,2$	$0,6$	$0,6$	2	?	2	?	$3,3$	$3,3$
α_{13}	2	2	2	2	?	$3,3$	$3,3$	2	$0,2$	$0,2$	$0,2$	$0,6$	$0,6$	2	?	2	?	$3,3$	$3,3$
α_{14}	$0,4$	$0,4$	$0,4$	$0,4$	?	$2,05$	$2,05$	$0,4$	$2,05$	$2,05$	$2,05$	2	2	$0,4$	?	$0,4$	?	$2,05$	$2,05$
α_{15}																			
α_{16}	$2,05$	$2,05$	$2,05$	$2,05$	?	$0,2$	$3,85$	$2,05$	$0,2$	$0,2$	$0,2$	$3,3$	$3,3$	$3,85$	?	$3,85$	?	$0,2$	$0,2$
α_{17}																			
α_{18}	$2,05$	$2,05$	$2,05$	$2,05$	?	$3,85$	$0,2$	$2,05$	$3,85$	$3,85$	$3,85$	$3,3$	$3,3$	$2,05$	?	$2,05$	?	$2,05$	$2,05$
α_{19}	$2,05$	$2,05$	$2,05$	$2,05$	?	$3,85$	$0,2$	$2,05$	$3,85$	$3,85$	$3,85$	$3,3$	$3,3$	$2,05$	?	$2,05$	?	$2,05$	$2,05$