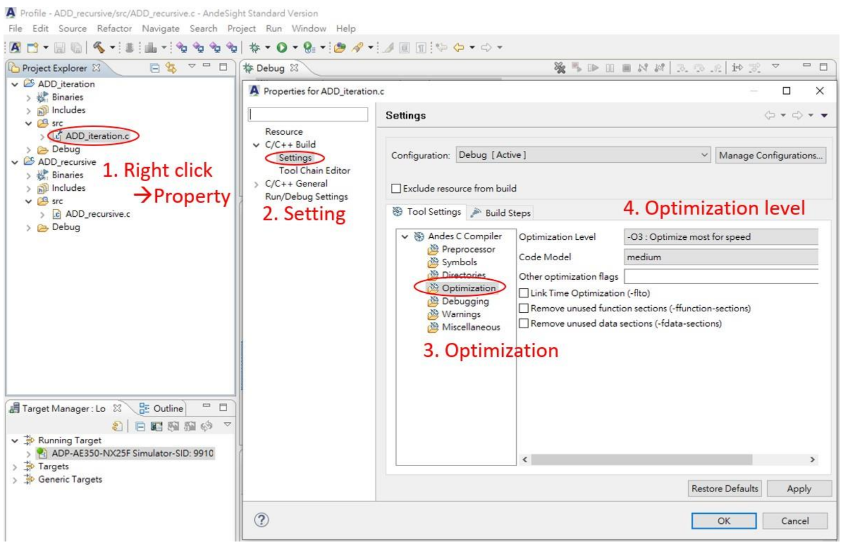
Department of Computer Science
National Tsing Hua University
EECS403000 Computer Architecture
Spring 2024, Homework 1
Due date: 2024/03/21

1. (40 points)
   Install and use AndeSight™ for RISC-V program development.
   (1) See AndeSight_STD_v5.3_Installation_Guide.pdf for the installation guide.
   (2) Create a new Andes C Project
      (i) Click on File New Project C/C++, and select “Andes C Project”.
      (ii) Create a project with the project name “fast_power_recur”.
      (iii) From Chip Profile, select chip profile “AE350” “ADP-AE350-NX25F”.
      (iv) From Project Type, select project type “Andes Executable” “Hello World ANSI C Project”.
      (v) From Toolchains, select the “nds64le-elf-mculib-v5d”.
      (vi) Other configurations are left as default.
   (3) Replace fast_power_recur.c in the project with the one we provided.
   (4) To build the project, click on the expanding arrow (a small triangle) beside “Build” in the toolbar “1 Debug” 1 Debug for project “fast_power_recur” in the toolbar.
   (5) To execute the program, press “Debug” in the toolbar “1 Application Program” and press “Resume” in the debug window.

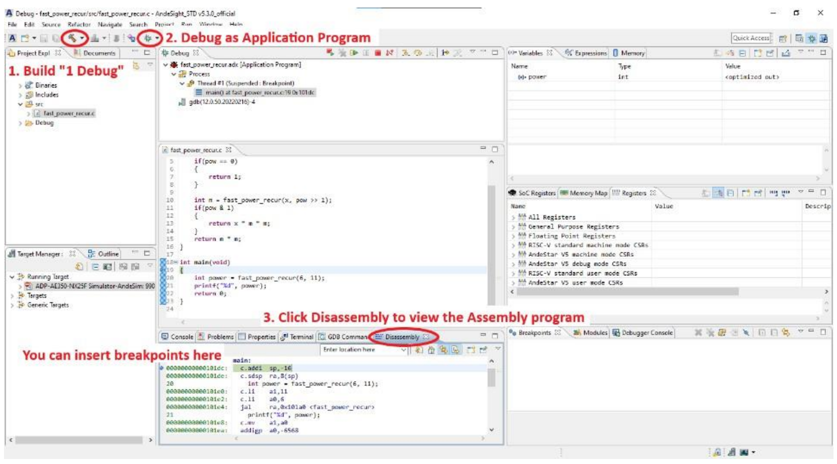
   You can follow the same steps for other program codes.

   To select (or check) the optimization setting, follow the figure below.



Configuration: Debug [Active]
Optimization Level: -O3 : Optimize most for speed
Code Model: medium
Other optimization flags:
☐ Link Time Optimization (-flto)
☐ Remove unused function sections (-ffunction-sections)
☐ Remove unused data sections (-fdata-sections)

   To inspect the assembly code of the program, follow the figure below.


1. Build "1 Debug"
2. Debug as Application Program
3. Click Disassembly to view the Assembly program
You can insert breakpoints here
Console
Problems
Properties
Terminal
GDB Command
Disassembly

```
fast_power_recur.adx [Application Program]
  Process
    Thread #1 (Suspended : Breakpoint)
      main() at fast_power_recur.c:19 0x101dc
  gdb(12.0.50.20220216)-4
```

```
    if(pow == 0)
    {
        return 1;
    }

    int m = fast_power_recur(x, pow >> 1);
    if(pow & 1)
    {
        return x * m * m;
    }
    return m * m;
}

int main(void)
{
    int power = fast_power_recur(6, 11);
    printf("%d", power);
    return 0;
}
```

```
                   main:
00000000000101dc:    c.addi  sp,-16
00000000000101de:    c.sdsp  ra,8(sp)
20                     int power = fast_power_recur(6, 11);
00000000000101e0:    c.li    a1,11
00000000000101e2:    c.li    a0,6
00000000000101e4:    jal     ra,0x101a0 <fast_power_recur>
21                     printf("%d", power);
00000000000101e8:    c.mv    a1,a0
00000000000101ea:    addigp  a0,-6568
```

In this exercise, we will experiment with the naïve and fast power computation in two different implementations (iterative and recursive) with AndeSight™. There are four source codes, namely naive_power_iter.c, naive_power_recur.c, fast_power_iter.c, and fast_power_recur.c. The optimization level -Og will be used by default in the following questions unless stated otherwise.

(a) (10 points) ***Effects of algorithms on performance***

Press "Profile" in the toolbar and select Profile as "Application Program". Press the button "Resume" in the debug window, record the CycC and InsC for the four functions listed in the table below and complete the table. Based on the characteristics of the programs, briefly compare and explain the differences between the naïve and fast power algorithms in their profiles.

naïve algorithm 是一項一項慢慢乘起來的，fast power algorithm 有點像是一組一組乘起來。

以 8 次方為例：

naïve algorithm: (((((((x * x) * x) * x) * x) * x) * x) * x) => 共做 7 次乘法

fast power algorithm: $((x^2)^2)^2$ => 共做 3 次乘法

=> fast power algorithm 的乘法運算次數約為 naïve algorithm 乘法運算次數取 $\log_2$

Function	Source	CycC	InsC
naive_power_iter()	naive_power_iter.c	89	50
naive_power_recur()	naive_power_recur.c	218	135
fast_power_iter()	fast_power_iter.c	42	27
fast_power_recur()	fast_power_recur.c	118	73

(b) (10 points) ***Effects of programming on performance***

From the table above and based on the characteristics of the programs, briefly compare and explain the differences between the iterative and recursive implementations of the fast power algorithm in their profiles. Suppose that they are executed in a processor with a clock rate of 3 GHz, what are the average CPI and CPU execution time for the fast_power_iter() and fast_power_recur() functions?

在這兩種方法乘法做的次數是相同的，但是遞迴的方法會需要 store 和 load 上一層函式的位址，因此遞迴的方法會花比較多 cycles。

Function	Average CPI	Average Execution Time
fast_power_iter()	1.56	14 ns
fast_power_recur()	1.62	39.3 ns

(c) (10 points) ***Effects of compilers on performance***

Compile fast_power_iter.c and fast_power_recur.c with two optimizations levels, -O0 and -O1. Record the CycC and InsC and compute their corresponding CPI for the two different optimization levels. Furthermore, briefly compare and explain the differences in their profiles.

在 Optimization level -O1 的 assembly code 裡面，比起 Optimization level -O0 的 assembly code 少了許多從記憶體 load 資料到 register 的指令，應該是透過某些優化方式，讓經常用到的 data 留在 register 裡面以減少 instruction 的數量。

Function	Optimization level -O0			Optimization level -O1		
	CycC	InsC	CPI	CycC	InsC	CPI
fast_power_iter()	191	87	2.20	42	27	1.56
fast_power_recur()	267	167	1.60	118	73	1.62

(d) (10 points) ***Compilers versus hardware implementations***

If we want to run the -O0 codes compiled in (c) on a faster processor to achieve the same speedup as running the -O1 codes on the original processor in fast_power_iter() and fast_power_recur(), what will the clock rates of the faster processor be for fast_power_iter.c and fast_power_recur.c respectively?

Denote clock rates as R

fast_power_iter.c: $R_{\text{faster processor}} = R_{\text{original processor}}$ * (191 / 42) = 3 GHz * (191 / 42) ≒ 13.7 GHz

fast_power_recur.c: $R_{\text{faster processor}} = R_{\text{original processor}}$ * (267 / 118) = 3 GHz * (191 / 42) ≒ 6.8 GHz

2. (25 points) ***Benchmarking***

Below is a comparison between three mobile phones and their processors.

Product	Samsung Galaxy S24 Ultra	Apple iPhone 15 Pro Max	Google Pixel 8 Pro
SoC	Snapdragon 8 Gen 3	Apple A17 Pro	Google Tensor G3
Cores	8 (1+3+2+2)	6 (2+4)	9 (1+4+4)
PDF renderer	227.4 Mpixels/sec	178.5 Mpixels/sec	153 Mpixels/sec
HDR	238.2 Mpixels/sec	232.4 Mpixels/sec	136.9 Mpixels/sec
Background blur	26.7 images/sec	27.9 images/sec	15 images/sec
Photo processing	64.9 images/sec	79.1 images/sec	47 images/sec
Ray tracing	7.38 Mpixels/sec	7.58 Mpixels/sec	4.52 Mpixels/sec

The information above is provided by https://nanoreview.net/en/soc-list/rating.

(a) (5 points) Follow the link https://nanoreview.net/en/soc-list/rating and fill in the table below.

Core name	Peak frequency of the most performant block of cores (MHz)			
Snapdragon 8 Gen 3	One core	Three cores	Two cores	Two cores
	Cortex-X4	Cortex-A720	Cortex-A720	Cortex-A520
	3300	3150	2960	2260
Apple A17 Pro	Two cores		Four cores	
	Everest		Sawtooth	
	3780		2110	
Google Tensor G3 Pro	One core	Four cores	Four cores	
	Cortex-X3	Cortex-A715	Cortex-A510	
	2910	2370	1700	

(b) (10 points) Suppose that we run three computer graphics and multimedia programs on all three smartphones:

Program A: Renders 114,000,000 pixels when viewing HW1.pdf.

Program B: Blurs the background of 2,000 images in the image gallery.

Program C: Processes 4,000 images in the image gallery.

For simplicity, we assume that the program only runs on a single core. The Samsung Galaxy S24 Ultra uses Cortex-X4, the Apple iPhone 15 Pro Max uses Everest, and the Google Pixel 8 Pro uses Cortex-X3. Furthermore, there is no other overhead. We are interested in the execution time (in seconds) and the clock cycles (in millions) of each smartphone. Use the provided information in the table and your answer in (a) to complete the table below.

Smartphone	Program A		Program B		Program C	
	Seconds	Clock Cycles	Seconds	Clock Cycles	Seconds	Clock Cycles
Samsung (Cortex-X4)	0.50	1650	74.91	247,203	61.63	203,379
Apple (Everest)	0.64	2419.2	71.68	270,950.4	50.57	191,154.6
Google (Cortex-X3)	0.75	2182.5	133.33	387,990.3	85.11	247,670.1

(c) (10 points) We are interested in comparing the performances of the three smartphones. Calculate the relative performance of the three smartphones, with each phone as the reference for comparison. Use your answer in (b) to complete the table below and summarize the performance results by calculating the **geometric mean of the performance ratio** of the three benchmark programs (Program A, Program B, and Program C). **Hint**: you might only need to compute some of the six values from scratch.

S24 Ultra 與 iPhone 15 Pro Max 效能相近，Pixel 8 Pro 效能明顯比較差。

Reference	Performance Ratio		
	Samsung Galaxy S24 Ultra	Apple iPhone 15 Pro Max	Google Pixel 8 Pro
Samsung Galaxy S24 Ultra	1	0.998	0.647
Apple iPhone 15 Pro Max	1.002	1	0.648
Google Pixel 8 Pro	1.545	1.542	1

3. (10 points) ***Performance and speedup***

   Assume that a program requires the execution of 100×10^6 FP instructions, 140×10^6 INT instructions, 110×10^6 L/S instructions, and 55×10^6 branch instructions. The CPI for each type of instruction is 3, 2, 5, and 3, respectively. Assume that the processor has a 5 GHz clock rate.

   (a) (5 points) By how much must we improve the CPI of INT instructions if we want the program to run two times faster? Please show the calculation procedure.

   FP: $300 * 10^6$ clock cycles
   INT: $280 * 10^6$ clock cycles
   L/S: $550 * 10^6$ clock cycles
   Branch: $165 * 10^6$ clock cycles
   (300 + 280 + 550 + 165) / 2 = 647.5 > 280
   Impossible

   (b) (5 points) By how much is the execution time of the program improved if the CPI of FP instructions is reduced by 28%, the CPI of INT instructions is reduced by 32% and the CPI of L/S instructions is reduced by 61% and the CPI of branch instructions is reduced by 64%? Please show the calculation procedure.

   (300 * 0.72 + 280 * 0.68 + 550 * 0.39 + 165 * 0.36) / (300 + 280 + 550 + 165) ≒ 0.525
   1 – 0.525 = 47.5%
   The execution time is reduced by 47.5%

4. (15 points) ***Amdahl's law and the eight great ideas of computer architecture***

   One of the great ideas of computer architecture is parallelization. Amdahl's law can be used to calculate the overall speedup of parallel executions. Amdahl's Law is defined as follow:
   where

$$S_{latency} = \frac{1}{(1-p)+\frac{p}{s}}$$

   $S_{latency}$: the theoretical speedup of the execution of the whole task,
   s : the speedup of the part of the task that benefits from improved system resources,
   p : the proportion of the execution time that the part benefiting from improved resources originally occupied.

   The ideal speedup of a parallelized program is the number of processors used. However, the theoretical speedups have limitations by the percentage of the application that cannot be parallelized, which includes the communication costs. The problem is that the communication costs are not fixed but often vary based on the number of processors used. In the following, let us consider the communication costs separately from the non-parallelizable execution of the program.

   (a) (5 points) Suppose we have a method to parallelize the fast_power_iter() function in (1) using an arbitrary number of processors. Moreover, the execution time T on one processor is the result obtained in (1)(c). Compute the parallel execution time of fast_power_iter() on 2, 4, 8 processors assuming 75% of the function is parallelizable and there is no communication cost.

   2 processors: 0.25 * 14 + 0.75 * 14 / 2 = 8.75 ns
   4 processors: 0.25 * 14 + 0.75 * 14 / 4 = 6.125 ns
   8 processors: 0.25 * 14 + 0.75 * 14 / 8 = 4.8125 ns

   (b) (5 points) Assuming the communication costs are 4% of the original execution time regardless of the number of cores, what is the speedup with 8 cores when 75% of the program is parallelizable?

   Speedup = (14 * 1.04) / (0.25 * 14 * 1.04 + 0.75 * 14 * 1.04 / 8)
   = (14 * 1.04) / (4.8125 * 1.04) ≒ 2.909

(c) (5 points) Assuming the communication costs are increased by 2% of the original execution time every time the number of processors is doubled, what is the speedup with n cores when 75% of the program is parallelizable? Furthermore, what is the specific speedup value when n = 8 in this scenario?

Let $k = \log_2 n$

Speedup with n cores = $(5.39 * 1.02) / (0.25 * 5.39 * 1.02 + 0.75 * 5.39 * 1.02^k / n)$

Speedup with 8 cores = $(5.39 * 1.02) / (0.25 * 5.39 * 1.02 + 0.75 * 5.39 * 1.02^3 / 8) ≒ 2.877$

5. (10 points) ***Integrated circuit cost and manufacturing***

Assume that a 50mm diameter-wafer has a cost of $9 and contains 95 dies. The yield for this wafer is 90%.

(a) (4 points) Find the defects per area for this wafer using the Equation on Page 28 of the textbook (or Page 28 of the slide).

$(1 / 3) * 2 / ((25^2\pi) / 95) ≒ 0.00523$ defects / mm^2

(b) (2 points) Find the cost per die for this wafer.

9 / (95 * 0.9) ≒ 0.105 dollars / die

(c) (4 points) If the number of dies per wafer is increased by 10% and the defects per area unit increases by 25%, find the new die area and new yield.

New die area = $(25^2\pi) / (95 * 1.1) ≒ 18.789$ mm^2

New yield = $1 / (1 + (0.00523 * 1.25) * (25^2\pi / (95 * 1.1 * 2)))^2 ≒ 0.888 = 88.8\%$