Hadassah Academic College | **Department of Computer Science**

Course: Computer Architecture | **Semester B 2023-24 Year 3**

Course Instructor: Dr. Martin Land | **מרצה הקורס: ד"ר מרטין לנד**

Submission dates: **Part 1** **30.6.2024**

Part 2 **11.8.2024**

Final Paper — Submission in Groups of Three

The practical goal of the Computer Architecture course is to impart the following abilities:

1. Reading an ISA and understanding the characteristics of its assembly language,
2. Understanding the basic operations in a RISC assembly language and analysis of simple programs,
3. Understanding the fundamental concepts in the organization of computer hardware and methods of ISA implementation,
4. Analysis of system performance and performance enhancement in practical situations.

In this final paper, you will apply these abilities on a CPU not discussed in detail in the course.

Notice that this assignment is a final paper and not an exam. A few differences:

- Exam questions are generally identical for each student. For this paper, each group of students writes a different program and determines the appropriate questions to answer.
- An exam evaluates the ability to apply a defined set of abilities in a short time. In this final paper, you have enough time to learn new material based on the course and apply it to a practical problem.
- In an exam, points are "taken off" from a base of 100% for incorrect answers to defined questions. In this final paper, there may be various reasonable approaches to analysis of code execution and enhancement. A discussion that is correct in a basic way will receive a base of 80%. Analysis, enhancement, and presentation at a higher than basic level will receive some proportion of the remaining 20%.
- As in any academic writing, it is important to explain assumptions, compromises, trade-offs, doubts, and lines of thought. **Numerical results presented without details of the calculation that produced them will be treated as incorrect.**

General Instructions

The DLXv6 is a model CPU designed for educational purposes. It is a simplified version of MIPS 32, the architecture of a series of commercial RISC microprocessors. For the purpose of this paper, we may assume:

- The MIPS32 hardware (organization, resources, pipeline, instruction implementation, and so on) is identical to the DLXv6c. Thus, analysis of program execution for the MIPS32 is identical to analysis of operation of the DLXv6c.
- The MIPS32 differs from DLX in the scope of its machine language: The MIPS32 language is richer and its assembly language is extended beyond that of DLX. Nevertheless, there is no change in instruction types (transfer, ALU, memory, branch). The MIPS32 assembly language expands several familiar DLX operations.
- The MIPS32 assembly language also includes convenient macros, making its programming appear more similar to Intel x86. The MIPS32 assembler converts these macros to standard MIPS32 machine language instructions (one or more instructions per macro).

As a commercial CPU, many advanced software tools are available for the MIPS32, including a cross-compiler (for example, versions of gcc that run on Intel x86-based PCs but output assembly code for MIPS32).

For this paper, you will extend your knowledge of DLX and learn commercial MIPS32 programming. Similarly to the exercises in the Architecture course, you will analyze the code structure of compiler output and the executions stages of the program in the CPU. In addition, you will edit the assembly code to enhance the run time.

The document **MIPS32 for DLX Programmers** is available on the course website. It summarizes MIPS32 instructions, code, data organization for programs based on the C language, and the most common macros.

Part 1 of the final paper must be submitted as a typewritten text-only file (no scans or photos of handwriting) in the moodle submission box before 23:59 on **30.6.2024**. **The filename must include the ID numbers of each member of the group**.

Part 2 of the final paper must be submitted as a typewritten (Office, PDF, or similar — no scans or photos of handwriting) in the moodle submission box before 23:59 on **11.8.2024**. **The filename must include the ID numbers of each member of the group**.

Paper contents

Part 1

A. Write a short C program with the following characteristics:

1. Defines variables of type **`int`** and writes to each one in the program,
2. Defines arrays of type **`int`** and writes to each one in the program,
3. Contains at least one loop,
4. In each loop iteration, reads variables/arrays, performs integer ALU operations, and writes the results,
5. Optional: defines and calls functions.

Note: The program must be in C, without C++ extensions or calls to standard libraries (such as **`stdio.h`**).

In standard C, execution begins with the first function defined — therefore **`main`** must be the first function defined.

An appropriate size for a program is a bubble sort implementation on an array of 10 items, or the multiplication of two matrices. More "interesting" programs are very likely to be more difficult to analyze. **Do not use bubble sort for your program**.

B. Download two programs for Windows from the course website:

1. **`cc1.exe`** — a simple C cross-compiler to MIPS assembly language
2. **`cygwin1.dll`** — provides a Linux runtime environment for the compiler

Save both programs in a folder together with the C program you have written.

At the command line (by running **`cmd.exe`** in Windows) run the following command:

`cc1 my_code.c` (where my_code.c is whatever name you gave your program).

The compiler will produce a file called **`my_code.s`** that contains the compile output in MIPS32 assemble language.

The command **`cc1 -h`** provides help for the compiler (not necessary for this assignment).

C. Submit the C program and the compiler output in two textfiles (the files my_code.c and my_code.s are text files). A **text file** contains only printable alphanumeric characters, without control characters. Word and PDF files are **not text** files. If you are not sure, open your files in Notepad.exe — if you can read the contents, then it should be all right.

Part 2

Write a paper (in the style of an article or a seminar paper) that conforms to the rules of academic writing. The paper should contain, at least, the following sections:

1. A cover page,
2. Your C program with a short explanation,
3. The compiler output, with each line numbered,

4. A description of the principal sections of assembly code, for example, the code before main(), the code that builds the data frame, control structures for building loops, closing the program, and so on,
5. Rewritten code translating the macros into pure MIPS32 assembly language,
6. Analysis of the execution stages of the assembly code (after step 5): instruction type distribution, identification of stalls, the number of clock cycles required to run the program, and so on,
7. Optimization: rewritten code (after step 5) that reduces the number of stalls and shortens run time,
8. Repeat of step 6 for the optimized code, and computation of the relative speedup in run time,
9. Decomposition of the program into threads, repeat of step 6 and computation of relative speedup,
10. Bibliography.

Note: Some complicated code structures are required in assembly code to conform with the C language. In performing optimization, it is not necessary to preserve these structures.

For a basic overview of steps 6 – 8, see exercise 9 andthe old exams. Any discussion of additional techniques from the course relevant to the program are welcome.

Some important requirements

1. In standard C and especially for the compiler **`cc1`**, the program will execute in the order that functions appear in the C code. For example in the program

```
void function(int A){
      int i ;
      for (i = 0 ; i < 5 ; i++) {
            A[i] = 2 * A[i] ;
      }
}
int main() {
      int A[] = {1,2,3,4,5} ;
      function(A) ;
      return 0 ;
}
```

the code for **`function`** will be written into the executable file before the code for **`main`** and the program will run in that order. As a result, the array **`A`** will not be initialized when first read. Also the data frame will not be built correctly. For this assignment, it is important that **`main`** appears first in the C program. Modern C/C++ compilers solve this problem for the programmer, but standard C for embedded systems has stricter rules.

2. In translating macros, leave the macro as a comment for **easy comparison**. For example, in translating **`move`**, write

```
27    add $21, $22, $0  // move $21, $22
```

3. In section 6, indicate each type of stall, for example

```
16    lw $22, 52($4)            LOAD
17    add $21, $21, $22         ALU ⟶ 1 CC stall
```

4. When optimizing the program, make sure to indicate changes so that they are **easy to identify**. For example, present the changes in a line-by-line comparison, preserving original line numbers:

Before

15	lw $21, 48($4)
16	lw $22, 52($4)
17	add $21, $21, $22
18	sw $21, 80($4)
19	lw $21, 56($4)
20	lw $22, 60($4)
21	sub $21, $21, $22
22	sw $21, 84($4)

After

15	lw $21, 48($4)
16	lw $22, 52($4)
19	lw $23, 56($4)
20	lw $24, 60($4)
17	add $21, $21, $22
18	sw $21, 80($4)
21	sub $23, $23, $24
22	sw $23, 84($4)