Practical Module 2: Register Transfer Language

This document contains the background information and exercises for the second module of the 2IC30 Computer Systems practical.

Session: 4

Introduction

In the previous module you experienced what it is like to build combinational and sequential circuits. During the course lectures, you have seen that once we can build memory blocks (registers) and logic blocks that perform calculations, we can start combining those blocks to form a microprocessor architecture capable of executing machine code instructions.

To do this, we first model a microprocessor architecture by building up the *Basic Data Path*. 'Instructions' are then executed by moving data around this data path and executing operations. This is expressed using a *Register Transfer Language*. Finally, you will see this Basic Data Path can be manipulated to allow a greater range of values to be processed, and how minor changes to the path increase the variety of instructions it is able to execute.

To complete this part of the practical successfully, you will need to understand the concepts of a processor architecture. You will need to think about how and when data (of which instructions are a part) is fetched and moved through a processor architecture, and how extending that architecture can make the execution of instructions more efficient.

To help you in this study, a software interpreter is provided that calculates the post-conditions of short Register Transfer Language (RTL) programs. The interpreter also gives some feedback on common mistakes regarding the mapping of instructions.

Session Schedule

Session	Preparatory Reading	Exercises
4	Read Chapter 4, Read 'Towards the Basic Data Path' Appendices A,C of textbook	4.1 Instruction Set on the Basic Data Path 4.2 Long Values, Addresses, and Displacements on the Basic Data Path 4.3 Instruction Set on the Extended Data Path(Prefetching)
5 (hour 1)		Automatically graded assessment

Top

Assessment Grading and Submissions

There are two elements to assessment of this module:

1. **An automatically graded set of RTL exercises submitted to MOMOTOR**
   This is worth **50%** of the module grade.
   You may work with your lab partner to produce the answers
   You **must** make an individual submission. This submission may be the same as your partner's.
   No submission means no grade (your partner's grade will not be allocated to you).
   The deadline for submission is one week after Session 4 (typically the day before Session 5).

2. **An automatically graded assessment of your understanding of processor architecture and low level programming fundamentals.**
   This is worth **50%** of the module grade.
   The test takes place during the first hour of the 5th practical session and will be taken under normal exam conditions. You must take the test on campus in the room scheduled for the practical.
   Further details are given at the end of this document.

Prior to (and during) Session 4

Work through the questions in 'Towards the Basic Datapath'. *These are not graded and solutions are not provided* (unless you approach the tutor). These questions will help you familiarise yourself with the architecture and offer an alternative perspective on the contents of Chapter 3 and 4 and Appendices A and C of the course text book. You can also begin working on the RTL exercises below and check out the practice test on ANS. Answer the guiding questions and keep your answers handy. *Attempt the practice quiz (available on ANS)* and note the feedback given, especially with regard to the variety of questions you may be asked.

Do not assume that the graded assessment is the same as the practice quiz questions with 'different numbers'.

Bring any questions you have regarding the RTL topic, the exercises and the practice quiz to the session.

The following pages contain details of the RTL exercises you need to complete.

Top

RTL File Syntax and Output

An example RTL sequence is provided for you (`example_fetch.rtl`). Shown below, this sequence models the fetch of an instruction into the Instruction Register (IR):

```
{Student1 Name: Number / Student2 : Number}
{IP = ip}
SigmaA, IP <- IP(Bbus), IP(Abus) + 1
{SigmaA = ip, IP = ip + 1}
SigmaR <- RAM[SigmaA]
{SigmaA = ip, SigmaR = RAM[ip], IP = ip + 1}
IR <- SigmaR(Bbus)
{SigmaA = ip, SigmaR = RAM[ip], IP = ip + 1, IR = RAM[ip]}
```

The text between { } is a comment. Comments have no effect on the evaluation of the code but should include any descriptive text you need to track the contents of all the relevant registers.

Any machine code instruction can be expressed as an RTL sequence that leaves the processor in a particular Postcondition, e.g. in the above sequence we wish that the IR contains an instruction copied from RAM. We provide you with an RTL interpreter that shows you the postconditions after an RTL sequence (in a raw text files) has been executed. Details of the interpreter are provided below, but you can see that executing the `example_fetch.rtl` sequence through the RTL interpreter will yield the following output:

```
> java -jar RtlTester.jar -s example_fetch.rtl

SIMPLE ARCHITECTURE
================================================================

Precondition:
SigmaA : sigmaa
SigmaW : sigmaw

SigmaR : sigmar
IP : ip
IR.val : ir.val
IR.compl : ir.compl
RA : ra
RB : rb
CC : cc
IR : ir

Program:
0. SigmaA, IP <- IP(Bbus), IP(Abus) + 1
1. SigmaR <- RAM[SigmaA]
2. IR <- SigmaR(Bbus)

Messages:

Postcondition:
SigmaA : ip
SigmaW : sigmaw
SigmaR : RAM[ip]
IP : 1 + ip
IR.val : ir.val
IR.compl : ir.compl
RA : ra
RB : rb
CC : cc
IR : RAM[ip]
```

In the output of the interpreter, the part under the header "Messages" is empty, meaning that in this case the RTL sequence contains no errors for the selected architecture.

The RTL interpreter (installation and use)

The RTL interpreter (`RtlTester.jar`) is distributed as a Java executable JAR file and is available for download from the Canvas assignment page along with an example `.rtl` file. Create a working directory to store your RTL files. The `RtlTester.jar` file needs to be placed in this directory as well. To run the interpreter, you need to have Java version 8 or later installed. When you have written an RTL sequence, e.g. in the file `file.rtl`, you can check your work with the following command in a terminal/command prompt[1]:

```
java -jar RtlTester.jar -s file.rtl
```

Running this command will provide you with the following output:

- The initial contents of the registers (precondition)
- The parsed RTL sequence from file.rtl
- Errors in the RTL sequence, if any
- The contents of the registers after executing the RTL sequence (postcondition)

The `–s` flag tells the interpreter to evaluate the RTL sequence on the simple data path architecture (see Figure 2 Simple Processor : The Extended Datapath_Figure 1 Simple Processor : Data PathFigure 1 Simple Processor : Data Path).

Replace `–s` with `–e` to evaluate the sequence on the extended architecture (see Figure 2 Simple Processor : The Extended Datapath).

Workflow

While solving the exercises of 4.1 to 4.3, you should include a comment on the first line with your student name and number. Comments should also be added to cover the pre- and post-conditions and other descriptions in your code. In other words, keep track of the content of all relevant registers at every step of the program. Please leave those comments in place in the files you upload, so that we can follow your reasoning in case an exercise gives rise to manual inspection.

After you have created a program, run it through the interpreter and verify that:

- the post-conditions you hoped to get correspond with the output of the interpreter.
- all your bus-assignments are correct and that any ALU operation you use actually exist in the ALU presented in the course text/lectures.
- There are no reported error messages.

The instruction fetch sequence above will be the first part of any program you write on the basic data path – do not forget to include it! (The sequence only fetches the instruction into the Instruction Register; the actual process steps must also be added). Later, when you write RTL sequences for more complex architectures, the instruction fetch does not need to be included.

When you are happy with your solutions then you can upload your answers to MOMOTOR. MOMOTOR expects each exercise to be in a specifically named file. These filenames are given in the exercises. Further details are provided at the end of the exercises.

[1] Instructions on how to open a terminal or command prompt, and access the right directory to run programs differ per operating system. Google is your friend.

Top

4.1 Instruction Set on the Basic Data Path

In this set of exercises we use the Basic Data Path, as given in Figure 5.

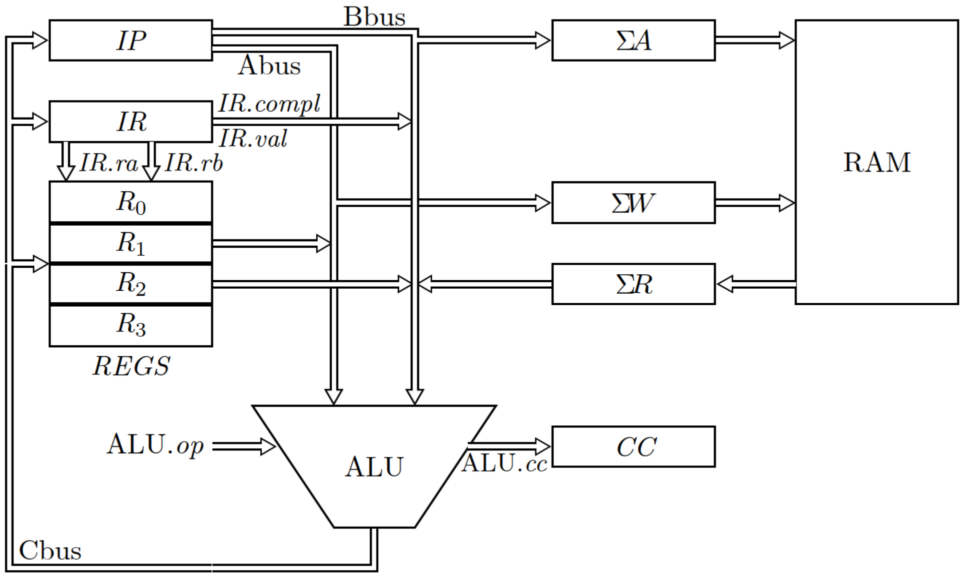


Figure 1 Simple Processor : Data Path

a) Describe for the following SP instructions how they can be executed by means of the Basic Data Path:

ADD R_i R_j	`exercise1.rtl`
LOAD R_i [*value*]	`exercise2.rtl`
BCC *disp* (when taken)	`exercise3.rtl`

Think!

- What addressing mode is being used for each of the above instructions?
- How many memory accesses (above those needed for the fetch) are required for each of the above instructions?
- Is there a relationship between the addressing mode and the number of memory accesses?
- Does the condition code of the branch have any effect on the RTL sequence?
- What is the value of *disp* relative to? Bear this answer in mind for subsequent exercises!

4.2 Long Values, Addresses, and Displacements

The Simple Processor has a word length of 16 bits; with 16 bits allowing 2^{16} combinations, we have 2^{16} = 65536 memory locations can be addressed. In view of this it is a pity that the binary instruction encoding of the simple processor allows for only 9 bits to encode values, addresses and displacements.

To remedy this situation we will modify the instruction encoding so that instead of using 9 bits to encode a value, an address or a displacement, we now use only 8 bits, and we use the ninth bit to indicate one out of two possibilities:

i) The ninth bit equals 0 : now the remaining 8 bits encode the value as usual; instead of v v v v v v v v v we now have 0 v v v v v v v v . We call this the *short form* of the instruction.

ii) The ninth bit equals 1 : now the remaining 8 bits are not used, and are, therefore, kept zero; the value is now encoded, in its full 16 bits, in the memory word directly following the instruction word proper. So, this case is encoded as `1 0 0 0 0 0 0 0 0`, and now the whole instruction comprises *two* 16-bit words: the first word contains the instruction with the new encoding, and the second word contains the operand value or address, in 16 bits. We call this the *long form* of the instruction.

So, in this arrangement, all instructions involving an operand (value, address or displacement) could have a short or long form depending on the operand value. If a processor implements both long and short form instructions then only values in the range [−128 . . +127] can be encoded in short form; for all values outside this interval the long form must be used. There is, however, a trade-off in being able to handle larger `value` operands.

To begin with understanding this, in the following exercise you can assume that a processor only implements long form instruction.

a) What is the RTL sequence for the long form of the `LOAD` R_i $[value]$ instruction?

`exercise4.rtl`

Think!

Unlike the above exercise if a processor has to handle both long and short form instructions then it only 'learns' if an instruction is in the long form *after* it has been clocked into IR, would this change your answer to the previous question?

If a processor can handle both long and short form instructions then how would an assembler that targets such a processor determine if a long or short form instruction is needed? Consider the range of values that can be encoded as operands by either form of the instruction.

Which instructions will use the `value` operand as an unsigned number? Which instructions will use it as a two's complement signed number? Does this affect whether a long or short form of the instruction is needed?

If a processor implements both long and form instructions what is the two's complement displacement, as used in branching instructions, relative to?

4.3 Instruction Prefetching

As explained during the Lectures and section 4.6 of the course textbook, (pre)fetching the next instruction while the current instruction is being executed is one way to optimise the throughput of instructions over time.

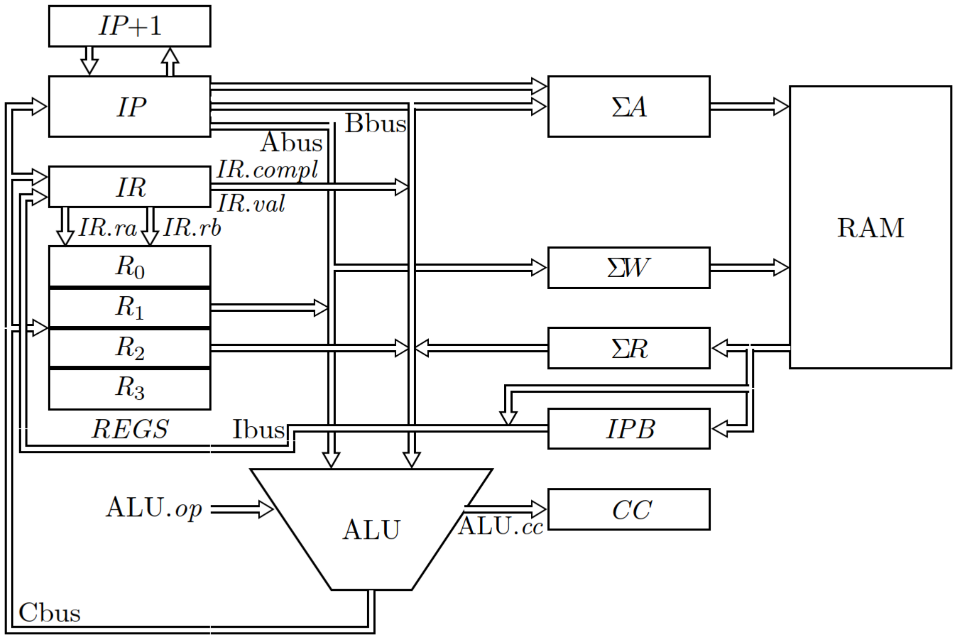


Figure 2 Simple Processor : The Extended Datapath

For the following questions you should implement the machine code instructions onto the Extended Data Path, as shown in Figure . You do not have to include this initial fetch, but you must ensure that when the sequence completes the processor is ready to immediately execute the next instruction (see textbook p.90 for an explanation of the assumed pre-conditions in the Extended Datapath).

a) Describe for the following instructions how they can be executed by means of the Extended Data Path:

ADD R_i *value*	exercise5.rtl
STOR R_i [*value*]	exercise6.rtl
BEQ disp (when not taken)	exercise7.rtl

b) Describe how the following long form instructions (see Long Values, Addresses, and Displacements), can be executed efficiently on the Extended Data Path:

ADD R_i *value*	exercise8.rtl
STOR R_i [*value*]	exercise9.rtl
BNE disp (when taken)	exercise10.rtl

Top

Momotor Submission

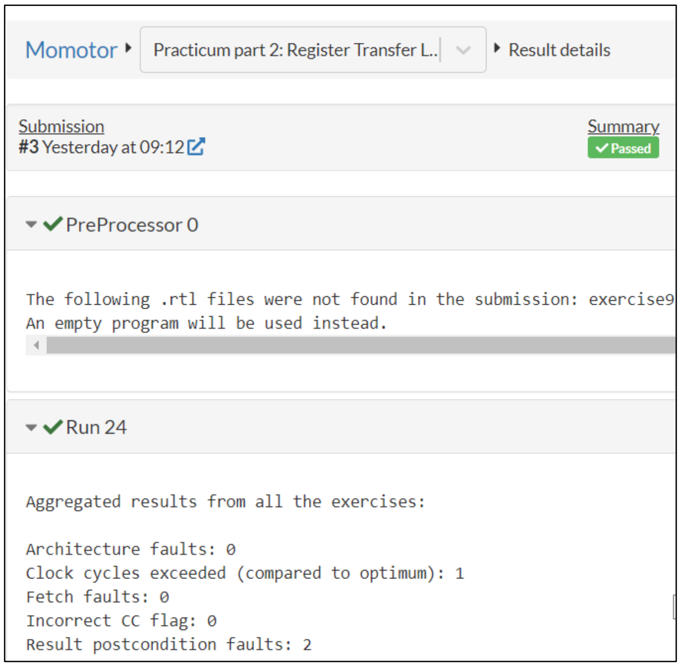
a. Each exercise must be in a separate `.rtl` file (the name is specified in the exercises).
b. Zip all the `.rtl` files into a file called `answers.zip`.
   (***DO NOT USE OTHER COMPRESSION FORMATS OR FILENAMES***)
c. Upload the zip file to the RTL MOMOTOR assignment on Canvas.
d. Select the MOMOTOR tab in the course navigation to see the results.

Plan your working time accordingly also noting that there is no MOMOTOR submission required *during* practical session 4. If you submit the same files as your lab partner then you are responsible for ensuring your submission is in the correct format.

You will only get 5 attempts to upload files on MOMOTOR. If you use more than this number of attempts (MOMOTOR is not able to limit uploads) your submission will be disqualified, and your grade for this submission will be a 1. Use the provided interpreter to check your solutions before attempting your first upload.
You must upload an individual submission for this work

As can be seen to the right MOMOTOR will verify if your instructions are correctly programmed according to the postconditions expected for that instruction, the number of clock cycles and whether there are faults in the use of busses and registers etc.

MOMOTOR will not give you feedback on each individual exercise. This is by design. By using the RTL Interpreter strategically alongside the course text book you will easily be able to determine which exercises have faults.



```
The following .rtl files were not found in the submission: exercise9
An empty program will be used instead.
```

```
Aggregated results from all the exercises:

Architecture faults: 0
Clock cycles exceeded (compared to optimum): 1
Fetch faults: 0
Incorrect CC flag: 0
Result postcondition faults: 2
```

Grading

You start with 30 points per exercise. Negative scores are not possible.

RTL file cannot be parsed/wrong name:	30 points penalty
Post Conditions are not as expected:	30 points penalty
Architecture Fault:	30 points penalty
Condition Code not assigned:	10 points penalty
Fetch Fault:	15 points penalty
Excess Clock Cycles:	10 points penalty per excess cycle

Top

RTL Test

The short test is open book but conducted under exam conditions.

- A laptop is required. It is your responsibility to ensure the laptop is in a condition to access and use the ANS platform (disable scheduled updates etc). Proctoring will not be used.
- You must be on campus in the practical room to take the test. No remote access will be permitted.
- During the test you will be allowed:
  A paper copy of the course text book (no online access to PDFs).
  Up to four A4 sheets of notepaper. You may write/print on both sides.
- No mobile phones, no headphones, no communication, no use of sources other than those listed above. No manual calculators - a calculator is built into ANS that you may use if needed.
- The vast majority of questions can be answered using material taken directly from the text book. It is advisable that you are familiar with how to locate information in the text book!
- *Mitigating circumstances as reported to your academic advisor, or in confidence with your tutor, are the only grounds whereby a test may be rescheduled.*

Questions are based on material in Chapter 4 of the course text book and the contents of this session. i.e.:

- The Basic Data Paths and variations on it (like the pre-fetch architecture).
- Short form instructions
- Long form instructions

You will not be asked questions relating to the instruction set extensions as mentioned in 4.7, 4.8, 4.9 and Appendix A of the course text book.

You may be asked to express numerical answers in binary, hexadecimal or decimal. You should be able to manually perform conversion between these number bases.

Top