TDTS08: Lab Report

Lab 4: VLIW Processors

Name	PIN	Email
Alexander Yngve	930320-6651	aleyn573@student.liu.se
Pål Kastman	851212-7575	palka285@student.liu.se

Contents

1 Introduction **3**

2 Method **3**

2.1 Basic Block 3

2.2 Dependencies 3

2.3 VLIW 4

3 Result **4**

4 Discussion **4**

1 Introduction

The purpose of this lab was to convert normal sequential code to VLIW instructions, so that we would get a greater performance. Basically we do what the VLIW compiler does during compilation time.

2 Method

The approach for this lab was the following:

1. Choose a basic block, and disassemble the block.
2. Find dependencies between the instruction in the block.
3. We pack the instruction into VLIWs.

2.1 Basic Block

To view all the basic blocks within the program (go.ss) the following command is issued:

```
vliwc /home/TDTS08/bin/go.ss | sort -nr +3 -4 | less
```

A basic block with at least 15 instructions is needed for the lab.

The next step is to view the disassemble the program to view the code for the basic block. This is done with the command below:

```
sslittle-na-sstrix-objdump -d /home/TDTS08/bin/go.ss | less
```

Which gives output similiar to this:

```
41b528:     28 00 00 00     lw $3,16($29)
41b52c:     10 00 03 1d
41b530:     28 00 00 00     lw $4,-31444($28)
41b534:     2c 85 04 1c
41b538:     a2 00 00 00     lui $6,4100
41b53c:     04 10 06 00
41b540:     43 00 00 00     addiu $6,$6,-6208
41b544:     c0 e7 06 06
```

2.2 Dependencies

To pack the instructions into Very Long Instruction Words, the dependencies between the instructions must be resolved. The dependencies of interest are the true data dependencies, read after write, where the output of one instruction is required as an input to one of the following instructions.

Output dependencies (write after write) and anti-dependencies (write after read) are not considered since the are artificial dependencies which can be resolved in preprocessing.

The true data dependencies should be visualized in a directed graph where the address of each instruction is a node. The edges between the nodes symbolizes a dependency. This graph should also be described in a textual representation which will be used by the *vliwc* program. An example of the graph file looks like this:

```
0x00000001
0x00000001 0x00000002
```

This file describes two instructions, *0x00000001* which is independent, and *0x00000002* which is dependent on *0x00000001*.

2.3 VLIW

The last step is to pack the sequential instructions obtained in section 2.1 into VLIW format with the help of the dependency graph from section 2.2.

The text format for the VLIW file begins with a line which specifies *alu_no*, *mul_no*, *fpu_no* and *bau_no* - how many ALUs, MULs, FPUs and BAUs the VLIW processor will have.

The next lines are the Very Long Instruction Words in the form of addresses to the sequential instructions. The first *alu_no* instructions will be ALU instructions, the next *mul_no* instructions will be MUL instrustions and so on.

An example VLIW file can look like this:

```
1           1   1   2
nop         nop nop 0x0041b528 0x0041b538
0x0041b540 nop nop 0x0041b530 nop
```

3 Result

4 Discussion

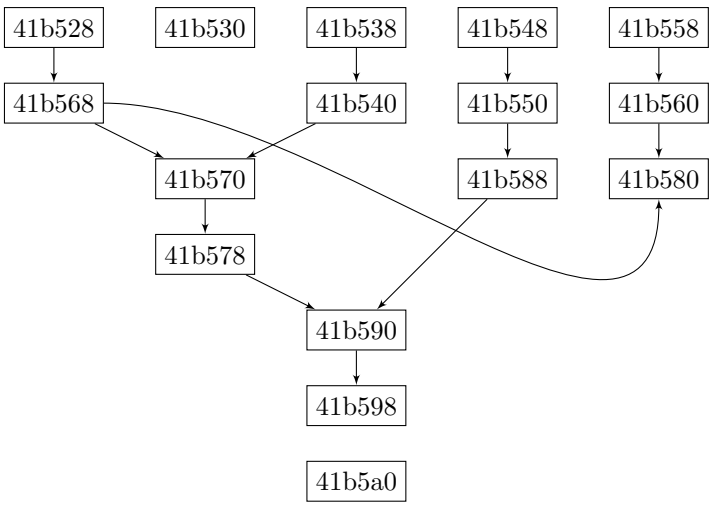

Figure 1 – Data dependencies between the instructions.