Writing Code for VLIW MIPS Architecture

Written by Kiwan Maeng

Collaboration Policy

Students must work on each question **alone**. Students may discuss high-level ideas only and **must not share any actual MIPS code** with each other.

Overview

In this project, students will write a simple assembly code for a MIPS single-cycle processor and a VLIW processor and run it on a functional simulator. The goal is to (1) review how to write a simple program in MIPS, (2) learn how a program can be accelerated with a VLIW architecture, (3) observe how different VLIW configurations can affect the performance, and (4) learn how a simple compiler optimization, i.e., loop unrolling, can improve the efficiency.

Part 0: Getting used to the infrastructure

A simple simulator for a single-cycle processor and VLIW processor is given. The simulator can be downloaded from Canvas and can be invoked by the following command:

```
python3 main.py --cpu-type <CPU_TYPE> --num-regs <NUM_REGS> --dm-size <MEMORY_SIZE> --mips-code <YOUR_MIPS_CODE_PATH> --issue-width <ISSUE_WIDTH>
```

Below summarizes the parameters:
--cpu-type: ‘singlecycle’ or ‘vliw’
--num-regs: The number of general-purpose registers. You can fix it to 32.
--dm-size: Size of the data memory in words. You can fix it to 65.
--mips-code: Specify the path of the MIPS instruction code you wrote
--issue-width: Issue width of the VLIW. Ignored if cpu_type==’singlecycle’

Our simple simulator only supports add, sub, lw, sw, beq, bne, addi, sll, srl, and nop, although adding more support is trivial. For VLIW, we assume the first half of the packet should be either ALU op or beq/bne, while the second half should be either lw or sw. Registers are all initialized as zero, and memory is arbitrarily initialized with an ascending number (memory address 0x0 holds 0, 0x4 holds 1, 0x8 holds 2, …).

Part 1: Writing a MIPS assembly that works

For this part, your code does not need to be heavily optimized, as long as it works and has some speedup after each optimization.

Q1-1: Write a MIPS program that is equivalent to the following C code. Assume that A starts from memory address 0x0. Your code should be actually looping 32 times using beq or bne (do not write a straight-line code with 32 sw instructions). See code1_0.txt for reference formatting.

```
int A[32];
for (int i=0; i<32; ++i)
  A[i] = i + 42;
```

Save your assembly code as code0_0.txt and try running it with the following command:

```
python3 main,py --num-regs 32 --dm-size 65 --cpu-type singlecycle --mips-code <PATH>/code0_0.txt
```

To check the correctness of your program, see if the printed memory state matches the following:

```
Final memory state: [42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64]
```

Q1-2: Write a MIPS program that unrolls Q1-1's code by an unrolling factor of 4. Save your code as code0_1.txt. Check the correctness of your code by running the same command with Q1-1 the new code. The memory state must be the same and the printed **# cycles at the end must be shorter than Q1-1**.

Q1-3: Write a VLIW code of Q1-1 by making instructions into packets for a 2-issue VLIW processor. Packet can be expressed by putting | between the instructions, as in the below examples:

```
add $1, $1, $2 | lw $3, 0($4)
addi $1, $1, 12 | nop
nop | sw $1, -4($5)
```

Save your assembly code as code0_2.txt and try running it with the following command:

```
python3 main,py --num-regs 32 --dm-size 65 --cpu-type vliw --mips-code <PATH>/code0_2.txt --issue-width 2
```

Final memory state must be the same and **# cycles must be less than Q1-1**.

Q1-4: Write a 2-issue VLIW code of Q1-2 by making instructions into a packet.
Save your assembly code as code0_3.txt and try running it with the command from Q1-3.
Final memory state must be the same and **# cycles must be less than Q1-2 and Q1-3**.

Part 2: Improving IPC

For this part, any code you write **must meet the cycle requirement**.

Q2-1: code1_0.txt holds a program that is equivalent to the following C code:

```
int A[32]; int B[33];
for (int i=0; i<32; i+)
  A[i] = B[i] + B[i+1]
```

This code performs the simplest form of a windowing average or convolution which is common in signal processing/machine learning. Try running the code with the following command:

```
python3 main,py --num-regs 32 --dm-size 65 --cpu-type singlecycle --mips-code
./code1_0.txt
```

Below should be your results:

```
# cycles: 193
Final memory state: [65, 67, 69, 71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99,
101, 103, 105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 32, 33, 34, 35, 36, 37,
38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61,
62, 63, 64]
```

Q2-2: Write a MIPS assembly that unrolls code1_0.txt by a factor of 4. Save it as code1_1.txt. When you run the code on our single-cycle simulator (using the command from Q2-1), the # cycles must be **121 cycles or less** for full credit.

Q2-3: Modify code1_0.txt for a 2-issue VLIW processor. Save it as code1_2.txt. # cycles must be **161 or less**, when running with the following command.

```
python3 main,py --num-regs 32 --dm-size 65 --cpu-type vliw --mips-code
<PATH>/code1_2.txt --issue-width 2
```

Q2-4: Modify code1_1.txt for a 2-issue VLIW processor. Save it as code1_3.txt. # cycles must be **73 or less**, when running with the command from Q2-3.

Q2-5: Modify code1_0.txt for a 4-issue VLIW processor. Save it as code1_4.txt. # cycles must be **129 or less**, when running with the following command.

```
python3 main,py --num-regs 32 --dm-size 65 --cpu-type vliw --mips-code
<PATH>/code1_2.txt --issue-width 4
```

Q2-6: Modify code1_1.txt for a 4-issue VLIW processor. Save it as code1_5.txt. # cycles must be **49 or less**, when running with the command from Q2-5.

Submission

Submit the 9 assembly files (code0_0--3.txt, code1_1--5.txt) by compressing it into .zip or .tar.gz and naming it with your psu id (e.g., kvm6242.zip). Upload it to Canvas.