Assignment 4

Due: 11:59 pm, Monday, April 2nd, 2018

Purpose: The primary purpose of this assignment is for you to i) become familiar with the micro-architecture of parallel processor, ii) learn about another parallel processor instruction set architecture, and iii) extend the functionality of an parallel processor emulator codebase.

Target Machines: For this assignment you will use any Linux computer running gcc version 4.8 with C++ 11 support. You may use any of the available ECE public servers (https://help.ece.gatech.edu/labs/names) for building and running your code or your own personal computer.

Assignment

Implement the complete instruction set for the mini-harp parallel processor as described in the next section. You could also refer to the Harp presentation for more information about the Harp micro-architecture. An emulator for mini-harp parallel processor has been provided with partial functionalities implemented and some sample programs compiled for the processor. Your goal is to complete the emulator by implementing the remaining instructions described by the ISA. The following should be completed:

1. General purpose register file
   - Implement the class RegFile in regfile.h.
2. Warp execution stage
   - Implement all remaining instructions defined in the ISA below ('I' column set to Y).
3. Performance counters
   - getInstructionCount(): return the total number of executed instructions.

Execution elements are as follows:

1. Use the software infrastructure provided to you. The software consist of the mini-harp emulator source code with Makefile.
2. The emulator takes as input the program binary and optional arguments for configuring the emulator.
   1. -r <regsize> : set the size of the register file
   2. -w <warps> : set the number of warps
   3. -t <threads> : set the number of threads per warp
   4. -o <file> : write the console output into a file
   5. -h : show command line usage
3. You have been provided three compiled programs for this assignment, together with their corresponding assembly source to use a reference as how the instructions are used:
   1. hello.bin : display Hello World! On a single-threaded warp

2. sum.bin : parallel sum of 4 arrays on using multi-threaded warp
3. barrier.bin : parallel sum of 4 arrays on using 4 synchronized warps

4. The output generated from running the emulator against a program will be captured into a file for validation.
5. Your programs will be tested on the PACE cluster with the following sequence of commands:
   - $ make
   - $ ./miniharp <program> -o result.log -r <regsize> -t <threads> -w <warps>
6. You can change any part of the infrastructure provided to you, including the input functions, data structures and makefiles.
7. Any additional files that must be included should be part of your submission.
8. You may not modify the build or execute commands used to run your program.

Mini-Harp Instruction Set Architecture

Machine Word:

- 32-bit

Instruction Encoding:

- The most-significant bit is 1 if the instruction is predicated and 0 otherwise.
- A single predicate register *@p0* is supported
- The next 6 bits are used for the opcode.
- Register operands are log 2 (#GPRs) bits long.
- Immediate fields are always the last field and occupy the remaining bits.
- Immediate fields are sign extended to the length of a machine word.

Assembly Language

It is RISC-like, and written destination register first (in this it differs from Unix assembly syntax). Registers names are prefixed with the percent sign (%) and predicate register names with the at symbol (@). Predicated instructions are prefixed with the predicate register name and a question mark: **e.g.** @p0 ? addi %r7, %r1, #1

Instruction Set

Instruction	**Description**	**I**
nop	No operation.	N

st %src0, %src1, #offset	Store %src0 into memory address %src1 + #offset.	Y
ld %dest, %src0, #offset	Load memory value at address %src0 + #offset into %dest	Y
ldi %dest, #imm	Load immediate.	Y
addi %dest, %src0, #imm	Add immediate.	Y
subi %dest, %src0, #imm	Subtract immediate.	Y
muli %dest, %src0, #imm	Multiply immediate.	Y
shli %dest, %src0, #imm	Shift left immediate.	Y
shri %dest, %src0, #imm	Shift right immediate.	
andi %dest, %src0, #imm	Add immediate.	Y
ori %dest, %src0, #imm	Or immediate.	Y
xori %dest, %src0, #imm	Xor immediate.	Y
add %dest, %src0, %src1	Add.	Y
sub %dest, %src0, %src1	Subtract.	Y
mul %dest, %src0, %src1	Multiply.	Y
shl %dest, %src0, %src1	Shift left.	Y
shr %dest, %src0, %src1	Shift right.	Y
and %dest, %src0, %src1	And.	Y
or %dest, %src0, %src1	Or.	Y
xor %dest, %src0, %src1	Xor.	Y
neg %dest, %src0, %src1	Two’s complement.	Y
not %dest, %src0, %src1	Bitwise complement.	Y
clone %src0	Copy all registers from active lane into specified %src0 lane. Register %src0 holds the destination lane index. e.g. If the register size per lane is 8, all 8 registers are copied.	Y
bar %src0, %src1	Synchronize %src1 warps with barrier identifier %src0. Register %src0 holds the barrier id (supported max value is	Y

	3). Register %src1 holds the number of active warps that should reach the barrier before execution resumes. e.g. bar %r0 %r1 where %r0=1 and %r1=2 will insert a barrier (#1) of size 2 to synchronize the first two active warps.	
wspawn %dest, %src0, %src1	Start new warp @ address %src0, copying %src1 into %dest	N
jmpi #addr	Jump to immediate (PC-relative).	N
jmpr %src0	Jump indirect.	N
jali %dest, #addr	Jump and link immediate.	N
jalr %dest, %src0	Jump and link indirect.	N
Jalis %dest, %src0, #addr	Jump and link immediate, spawning %src0 active lanes.	N
jalrs %dest, %src0, %src1	Jump and link indirect, spawning %src0 active lanes.	N
jmprt %addr	Jump indirect, enable single lane 0.	N
notp @dest, @src0	Inverse predicate value.	N
rtop @dest, %src0	Set @dest if %src0 is nonzero.	N
isneg @dest, %src0	Set @dest if %src0 is negative.	N
iszero @dest, %src0	Set @dest if %src0 is zero.	N
Halt	Stop warp execution.	N

Grading Guidelines

For your information here are the grading guidelines

- Program compile and executes without errors: 25 points
- Program executes *hello.bin* correctly: 25 points
- Program executes *sum.bin* correclty: 25 points
- Program executes *barrier.bin* correclty: 25 points

Submission Guidelines

All program submissions should be electronic. Submit a zip file with i) the complete software

infrastructure including files that were not modified, and ii) the PDF file of the report. The zip file should be named <last_name>.Assignment-4.zip. Submissions must be time stamped by midnight on the due date. Submissions will be via T-square.

Note: **No late assignments will be graded**. Remember, you are expected to make a passing grade on the assignments to pass the course!