Baseline Project Description
Due: Friday, 11:59 pm, April 27th, 2018

Goal: Implement the PDOM algorithm for branch divergence in GPUs.

Target Machines: 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.

Description

Extend your solution to assignment 4 by implementing the SPLIT and JOIN instructions for the mini-harp parallel processor as described in the next section. You could also refer to the Harp presentation for more information about branch divergence in 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.

Execution elements are as follows:

1. Use the software infrastructure for miniharp provided to you for Assignment 4.
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. Your programs will be tested with the following sequence of commands:
   - $ make
   - $ ./miniharp <program> -o result.log -r <regsize> -t <threads> -w <warps>
4. You can change any part of the infrastructure provided to you, including the input functions, data structures and makefiles.
5. Any additional files that must be included should be part of your submission.
6. You may not modify the build or execute commands used to run your program.

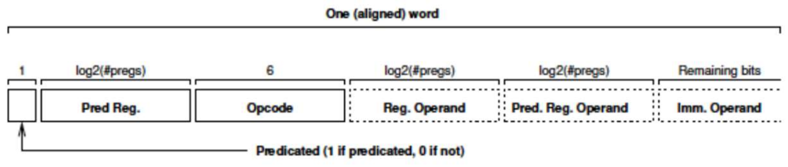
Mini-Harp Instruction Set Architecture

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**	**TODO**
Split	**Control flow divergence**	**Yes**
Join	**Control flow reconvergence**	**Yes**
Nop	No operation.	No
st %src0, %src1, #offset	Store %src0 into memory address %src1 + #offset.	No
ld %dest, %src0, #offset	Load memory value at address %src0 + #offset into %dest	No
ldi %dest, #imm	Load immediate.	No
addi %dest, %src0, #imm	Add immediate.	No
subi %dest, %src0, #imm	Subtract immediate.	No
muli %dest, %src0, #imm	Multiply immediate.	No
shli %dest, %src0, #imm	Shift left immediate.	No
shri %dest, %src0, #imm	Shift right immediate.	
andi %dest, %src0, #imm	And immediate.	No
ori %dest, %src0, #imm	Or immediate.	No
xori %dest, %src0, #imm	Xor immediate.	No
add %dest, %src0, %src1	Add.	No
sub %dest, %src0, %src1	Subtract.	No
mul %dest, %src0, %src1	Multiply.	No
shl %dest, %src0, %src1	Shift left.	No
shr %dest, %src0, %src1	Shift right.	No
and %dest, %src0, %src1	And.	No
or %dest, %src0, %src1	Or.	No

xor %dest, %src0, %src1	Xor.	No
neg %dest, %src0, %src1	Two's complement.	No
not %dest, %src0, %src1	Bitwise complement.	No
clone %src0	Copy current lane registers into specified %src0 lane.	No
bar %src0, %src1	Synchronize %src1 warps with barrier identifier %src0.	No
jmpi #addr	Jump to immediate (PC-relative).	No
jmpr %src0	Jump indirect.	No
jali %dest, #addr	Jump and link immediate.	No
jalr %dest, %src0	Jump and link indirect.	No
Jalis %dest, %src0, #addr	Jump and link immediate, spawning %src0 active lanes.	No
jalrs %dest, %src0, %src1	Jump and link indirect, spawning %src0 active lanes.	No
jmprt %addr	Jump indirect, enable single lane 0.	No
notp @dest, @src0	Inverse predicate value.	No
rtop @dest, %src0	Set @dest if %src0 is nonzero.	No
isneg @dest, %src0	Set @dest if %src0 is negative.	No
iszero @dest, %src0	Set @dest if %src0 is zero.	No
Halt	Stop warp execution.	No

Testing

Your implementation will be tested for correctness in the following cases:

- Divergent branches
- Nested divergent branches
- Divergent loops

No test cases will be provided. You are expected to write your own assembly language test cases to test your implementation. Provide 3 test cases along with your submission.

Submission:

Submit your entire project as a zip file. Include test cases.

Report

Submit a Project Report. With the following information

1. A description of your implementation of PDOM using the split and join instructions. Describe the specific data structures -use figures.

   Note that there are several subtle aspects of the implementation that have alternative solutions. For example, the communication of the reconvergence address. This can require additional support from other parts of the ISA. Your explanation must make this clear. Keep in mind that this is not a simulator – this will simplify the implementation.
2. Describe the test cases and their rationale. Your test cases should support multiple warps.
3. Limit your report to 2-3 pages.

Extra Credit

Extend your implementation to support Thread Block Compaction (TBC). In this case, your implementation need only be functional for a fixed number of warps and a warp size of 8. However, your implementation should be able to be recompiled to support 8, 16, or 32 warps Treat all the warps as belonging to a single TB.

The report should be extended to include the same three elements addressing the description of the TBC implementation. You are to provide your own i) tests cases (2), and ii) traces of activity masks and reformed warps that illustrate how TBC is working.

This extra credit report and implementation is worth an additional 35% of the project grade if you do the project alone. If you choose to team up, then you are required to do both the baseline project and TBC for full credit and will not receive extra credit.