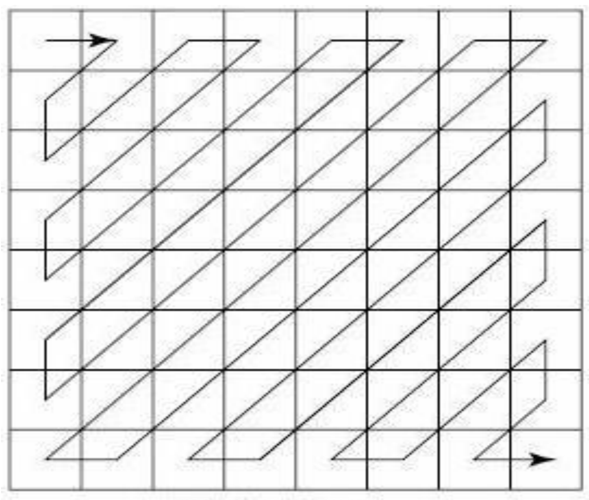
LABS 5-8

Objectives

1. Implement Full Search Based Variable Block Size Motion Estimation using MIPS ISA with the diagonal search pattern shown below (Page 1-4)



2. Implement datapath components (Page 5-9)

Objective 1 Logistics

- The template file **“vbsme.s”** includes an overview of the algorithm and public test cases.
- Demonstration due during your designated lab section on Lab8 day.
- Demonstration will include running your program using public test cases.
- A signup sheet will be available to choose your demonstration time slot at the beginning of the lab.
- Submit only the “vbsme.s” on D2L to the designated dropbox.
- Include the overall percent effort of each team member in the file.

Objective 1 Rules and Grading Requirements

- you must follow the given diagonal search pattern
- you are not allowed to use special registers in your implementation
    - "s" and "t" registers are the only registers to store values during the execution of the program
- avoid the following commands: division, mod

Objective 1 Scoring (250 points)

- Public test cases (70 points)
- Private test cases (180 points)

Objective 1 Assumptions

- Frame size : 16x16 to 64x64, where x and y dimensions can be any integer between 16 and 64 (can be square or rectangle)
- Window size : One of 7 specified dimensions in the "vbsme.s".

Objective 1 Strategy: You should approach the problem with three tasks:

- Task 1: implementation of the SAD subroutine that just computes the sum of absolute difference for a given window size.
- Task 2: implementation of the address generation for reading the corresponding frame elements based on the current position in the frame.
- Task 3: implementation of the scan pattern moves.

I highly recommend starting with the C (or your favorite language) based implementation for each task. It is even better if you implement in three address operand form in C. You should pass address of the first element for the window and the frame data. Then use window size as stride for accessing the subsequent rows of the frame data. Even though the test data seems to be a 2D array, to QtSpim it is a one-dimensional array. After confirming the SAD for a given position is working properly, then write the routine that realizes the required access pattern. Debugging becomes manageable with a modular design.

Although it is NOT mandatory in this lab, it is **highly recommended** that you only use the instructions given in Table 1 (see page 6) when implementing the SAD routine since they will be the only supported instructions on your MIPS Datapath design, which will be running your SAD algorithm in future labs.

Objective 1 Grading and Demo: Join the lab on the due date and signup for a demo slot (google sheet to be shared) for the demo.

- Demo will start with running your code with public test cases.
- All team members MUST be present during the demo for answering questions.
- During the lab session, TAs will give priority to demos. If you are still debugging your code, TAs will help only after taking all the demos.
- No submission is allowed after the demo. If done, then it will be accounted as a late submission and re-demo will be needed.

Objective 1: Offline testing with private test cases - When grading your "vbsme" implementation, we will use our vbsme.s template with private test cases. We will copy your code starting with the line that says "insert your code here" into the new template and run all test cases.

Objective 1 Penalty Conditions

- Percent effort not reported (20% penalty)
- Late submission or late demonstration (10% per day)
- Submitting files in a folder or in compressed form (zip/tar). (20% penalty)
- Changing the file name or extension. (20% penalty)
- Failing to demonstrate (90% penalty)
- Illegal register usage: max score 100pts,

- Illegal scan pattern usage: maximum score 50pts
- Unable to answer questions about your implementation during demo: maximum score 100pts
- Any form of plagiarism will result in score of 0 and disqualify the team from the competition.

Common Questions on SAD Routine

Q1: How are the frame and window matrices stored? Are they stored in sequential memory slots?

Answer: n the sum of array declaration is :
N: .word 3 # 'N' is the address which contains the loop count, 5
X: .word -2, -4, 7 # 'X' is the address of the 1st element in the array to be added
SUM: .word 0 # 'SUM' is the address that stores the final sum

in the memory data is stored sequentially 3 -2 -4 7 0
we use N, X , SUM to access the proper location.

Q2: Where to exit to? When we finish the final test, is there a specific location we are supposed to send the program? We are getting the attached error, which I'm assuming is because a return register is being set to 0. What I am not sure of is if this is purposeful or if this is because our program is rewriting that register to 0 before it is used.

Answer: If you are using JAL instruction then make sure to use $ra register to get back to proper retrun address. If you are not using the JAL then make sure to save return address on stack and use it to jump back. At the end of the program, you can either do as safe exit or go into an indefinite jump loop to same address. example:

```
loop: j loop
```

Q3. Will we ever have to worry about the size of the frame being greater than the size of the window?

Answer: The Frame should always be the same size as the window or larger.

Q4: In the word document, it says that we can assume the size of the frame will be between 16x16 and 64x64 where x and y can be any integer between 16 and 64. Test case 14 in the file vbsme.s has a frame size of 4x4. Will this test case be graded since it goes against the assignment description?

Answer: They (test0 and 14) were given so that students can use them in initial stage for quick validation of codes and debug purposes.

Q5. How do we access an index of an array based on a variable value that we cannot hardcode (for use in instructions like load word and store word)?

Answer:
You need to calculate memory location corresponding to array index you want to read using base address of the frame or window, and their dimensions.

Q6. Calculating SAD values
Is it sufficient to calculate the sum of the entire array for both the window and the frame and then take the absolute value of the difference? Or is it better to calculate the sum of difference in each indices since two different frame/windows can generate the same SAD value (even if they're not matching)?

Answer: Taking sum first and then difference is not correct. You need to take difference first and then sum. That's why it is called SAD: Sum of Absolute Difference.
Suppose you had the window:
1 0 0
0 0 0
0 0 0

And the Frame:
0 0 0
0 0 0
0 0 1

Taking the Sum of both Matrices, and then looking at their differences gives 0, while the SAD of this pair is actually 2.

Q7. Will all SAD values be 0?
I know the project says to find the minimum SAD value, but the test windows seem to have an exact match on the frames. Would it be better if we were to only look for the exact window?

Answer:
There may or may not be exact match.

Q8. Initializing Sum Abs Diff
Do we need to initialize the Sum of Abs Difference to a specific value? If you do not initialize it, then using a comparison for the minimum SAD will never execute because 0 will always be less than any computed SAD.

Answer:
You can initialize it to largest possible sad value or the first SAD value should be initialization. Assuming the SAD value between the first window and frame is not the largest possible value, the first SAD calculation would become the minimum anyway.

Q9. vbsme, negative numbers? I noticed that none of the frames or windows given in the test cases have negative numbers. Is it safe to say we don't have to worry about getting negative numbers in any test cases when we demo our vbsme code?

Answer:
Correct no need to worry about negative values

Objective 2 Datapath Component Design and Simulation (Not graded, no demonstration required)

Objective 2 Part 1:

- Below is a list of datapath components needed in your project. Templates are in the "DatapathComponents" folder.
  - RegisterFile
  - DataMemory
  - SignExtension
  - Mux32Bit2To1
- Follow the comments given in the source codes
- Synthesize and conduct functional verification with post-routing simulation for each component
- Exhaustively test your components
- It is your responsibility to make sure that these data path components are working properly. We will not be testing the functionality of these components individually
- No demonstration required for the datapath components
- You should plan to complete by Lab 4.

Objective 2 Part 2: ALU Design (**Start this exercise after covering ALU Controller Topic in the class**)

- Design and develop an **ALU** that supports all operations required by the given MIPS ISA.
- Template for the ALU is in the "DatapathComponents" folder.
- No demonstration required for ALU design.

Suggested ALU Design Approach

- Study the given instruction list (next page) and identify all operations needed. Refer to the MIPS ISA Reference.
  - **List** all the arithmetic operations.
- **Draw** the block diagram of the ALU module with inputs and outputs properly labeled with their bit-width.
  - You need to revise the ALU outputs of the ALU32Bit.v from "Part 1". You need to accommodate potential 64-bit outputs from certain instructions, as well as deal with interpreting the outcomes of arithmetic operations needed by different types of branch instructions. Overflow detection is not needed.
- Test your ALU in post-routing simulation for all the operations. There is no demonstration requirement for this task.

Table 1. Required MIPS Operations for the datapath design

Type	Instruction	Code	Type	Instruction	Code
Arithmetic	Add	add	Logical	And	and
	Add Immediate	addi		And immediate	andi
	Subtract	sub		Or	or
	Multiply	mul		Not or	nor
Data	Load word	lw		Exclusive or	xor
	Store word	sw		Or immediate	ori
	Store byte	sb		Exclusive or Immediate	xori
	Load half	lh		Shift left logical	sll
	Load byte	lb		Shift right Logical	srl
	Store half	sh		Set on less than	slt
Branch	branch if greater than or equal to zero	bgez		set on less than immediate	slti
	branch on equal	beq			
	branch on not equal	bne			
	branch on greater than zero	bgtz			
	branch on less than or equal to zero	blez			
	branch on less than zero	bltz			
	jump	j			
	jump register	jr			
	jump and link	jal			

Common Questions

Q1. When it says, "list all of the arithmetic operations", does it mean that we should determine if an instruction requires add, multiply, subtract, etc. or does it mean list all of the operations we will need to implement the full set of instructions? The reason this task seems confusing to me is because we already know we're going to have to handle add, multiply, subtract, and, or, and set on less than. I don't see what else we could really need in terms of arithmetic operations.

I'm also confused about what the block diagram of the ALU should look like. Are we expected to have different controls for each one of the branch instructions to determine how to interpret the result of the ALU? This seems like it should be outside of the ALU, so are we supposed to draw more of the datapath?

Answer:
Make sure to follow these steps:

1) Table in the lab description lists all the instructions that needs to be implemented. Read it carefully.
2) Refer to "MIPSInstruction" reference manual to see how all of these instructions are actually executed and what ALU operations are needed to execute all these instructions. Make a list of these operations.
3) Narrow down on operations (remove redundant operations). Identify all the unique operations.
4) Identify the width of the control signal you will need for your ALU to identify each of these unique operations.
5) Identify what all input and output signals you might need to add in your ALU (if needed) to implement these instructions. Adding new ports later in the project will need a lot of changes in your design, so it's better to do it early in the stage.
6) Make a block diagram of your ALU on a sheet of paper with required inputs (including control signal) and outputs (we are not looking for RTL designs). Note down all the operations that you will need to implement in ALU.

You need to identify all the operations you might need to include in your ALU for implementing all the instructions. For example, "slt" instructions will need to check for "less than condition" inside ALU and similar operator might be used by "bltz" depending on your implementation. Similarly, an "Add" operator can be shared among LW and ADD instructions.

Arithmetic instructions are different than operations that are discussed above. You have been asked to list all the unique operations you will need to implement in your ALU to include all the instructions in Table 1. Same operations can be shared between multiple instructions. READ MIPSInstruction manual and above responses carefully.

There are conditional branch instructions which will need ALU involvement. Further, you need to control "zero flag" inside your ALU to enable or disable branching. For Data instructions, you will need ALU to calculate the memory address you want to access in data memory. For instructions related to "move" you can use ALU to assign input to output.

Q2. In ALU32Bit.v, the outputs ALUResult and Zero were not declared as a reg type. I tried to assign these outputs values, and got an error "procedural assignment to a non register is not permitted". I declared both outputs as a reg type and the error disappeared. My question is, whether or not we are meant to declare the outputs as reg type, and if not, how are we supposed to assign values to the outputs?

Answer:
Components files given to you is just a reference. You can change the data types and add or remove ports if you think you will need that in your implementation.

Q3.I'm very confused for the instructions that involve the Hi and Lo registers. How do we access them when they are special registers and not in our register file component?

Answer:
Hi and Lo are two special registers outside the register file. each one is 32 bits wide.

when you multiply two 32 bit values you generate a 64 bit value.

higher order 32 bits and lower order 32 bits of the product are mapped onto the hi and loo registers respectively.

Q4. Can we read 4 registers from the regfile at once. Also can we write to to register in the regfile at once. Because we need to read hi/lo and also read 1 and read 2 at the same time, so that's 4. And then we need to write into hi/lo, so 2 writes. The only other solution I could think of was putting hi/lo into their own register file

Answer:
HI/LO registers are not the part of the register file. They are internal registers for ALU explicitly designed for special instructions. You will be designing these registers in execute stage where as registerfile (32 addressable registers) is a part of the decode stage.

Q5. M & DM: "Unconnected ports Address[0] Address[1] Address[9] ..." Synthesis Warnings
Why does the input Address for the memory modules have to be 32 bits? In DataMemory, we only use bits [8:2] for addressing and in InstructionMemory, we only use [11:2] for addressing? I am worried that vivado will not handle these unused Address bits well and I am wondering why we even need them in the first place to be passed to the memory units in the first place. Why cant we send only Address[8:2] into InstructionMemory and only Address[11:2] into DataMemory? If I leave it as is, will these unconnected ports prevent the code from being successfully implemented on the FPGA?

Answer:
Vivado will trim of unused logic and ports during synthesis and implementation phase. For example, if you try to connect "port1[31:0]" to "port2[11:0]". Vivado will trim of "port1[31:12]" and will connect "port1[11:0]" to "port2[11:0]"

Q6. Unroutable DSP cascade connection
I get the following error when trying to run implementation.

[Place 30-119] Unroutable DSP cascade connection found. Port 'PCOUT' of DSP block 'ALUResult0__1' can drive only 'PCIN' port of different DSP block.

Everything works perfectly in post synthesis simulation. Any idea why this is happening? I also have the warning from synthesis saying sequential elements are being removed(all pipeline register outputs) , but the simulation still works so I'm not really sure what to do about that

Answer:
Check if you are using blocking assignment for the Madd and Msub operations and remove blocking assignments.

Q6.1: Currently we only have it blocking because we parse multTemp(64 bits) to HI and LO, so multTemp needs to update immediately for that to work. How could we calculate the multiplication first without blocking?

Answer: you need to add the previous value of HI and LO registers with current result of multiplication for Madd. Basically, you need to have an input port HILO (which comes from the output of HI and Lo registers) for ALU, and used that to complete Madd and Msub operations. To avoid latches, make sure to give some default values to output and intermediate signals before entering into case or if-else statements.

Q7. Inferring Latch/ unused sequential elements
what does inferring a latch mean? I have my case statement with an output for every case I have written, yet it says I am inferring a latch. I do not have every case possible written but every case I have written has an output. Also I am getting a warning about unused sequential elements, however the element it claims is unused is an element that I have connected back to a mux, it is the PC value that can get shifted but doesnt in this lab. Any ideas?
Synthesis Warning "Ports drive by constant Zero"

Answer: If all the modules are properly connected then it should not be an issue. Also make sure that all the instruction memory locations are initialized to default values. Initialize all the intermediate signals. Have you tried to open the Schematic? The wire could be showing that it is hooked up and also running to ground. In such cases you need to initialize the wire in the top level design.