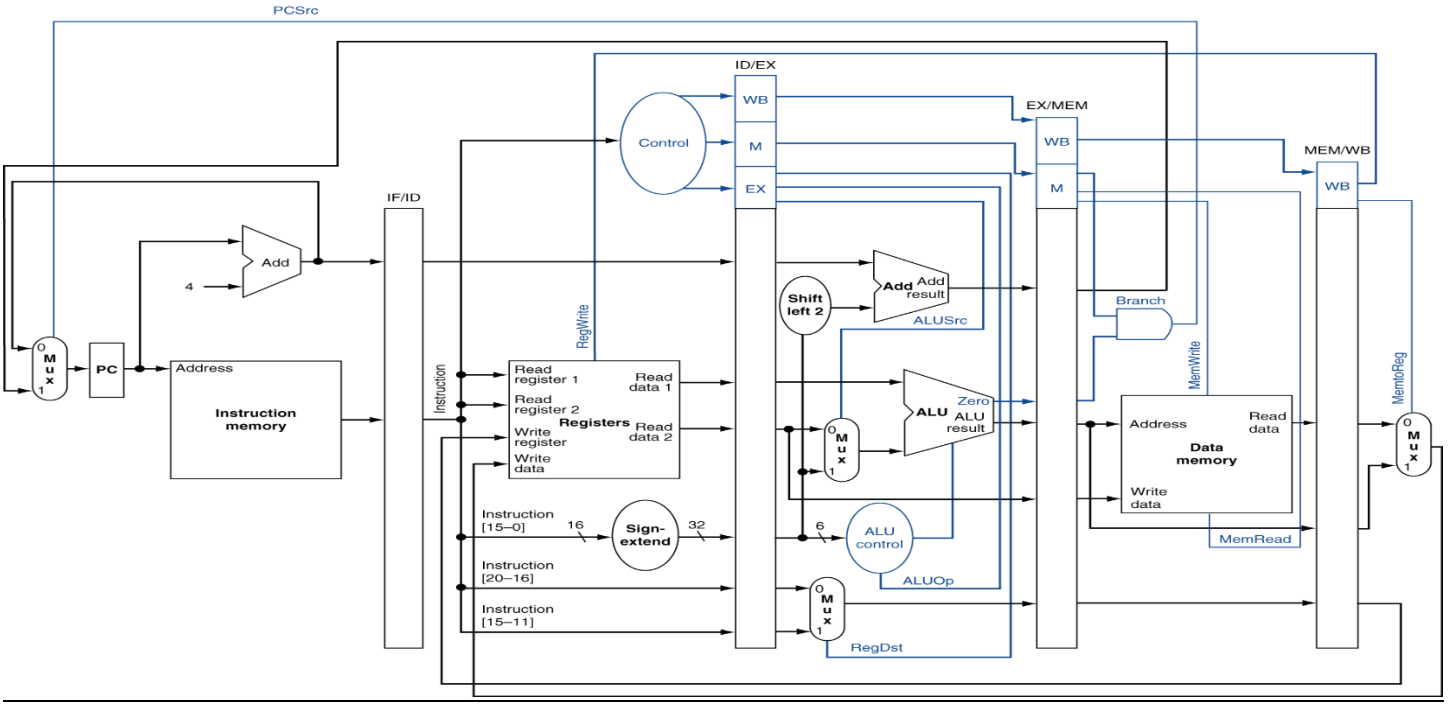
ECE369A Computer Organization
LABS 4 and 5

At this point you should have all your datapath components implemented including the ALU that supports the target ISA given in Table 1 from Lab3 document in page 6. Our objective is to design and build a datapath that is capable of executing all instructions in the target ISA. As you are implementing and debugging your datapath, you may want to check the "Common Questions" section of this document starting in page 9.

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			

Datapath



The datapath figure above is slightly different from the single cycle datapath we covered in the class. This datapath is actually a pipelined version. We will partition the execution of an instruction into 5 stages. These stages are:

1. Instruction Fetch (IF)
2. Instruction Decode (ID)
3. Execute (EX)
4. Memory Access (MEM)
5. Write Back (WB)

Between each stage we have a register file named as IF/ID, ID/EX, EX/MEM and MEM/WB. The size of the register file depends on the number of bits transferred from one stage to another. For example:

- IF/ID register is storing the 32-bit PC value and the 32-bit instruction read from the instruction memory.
- ID/EX register stores all the control signal generated by the Controller along with the two register values read from the register file, sign extended offset field, potential destination registers (Rd – I[15:0] and Rt – I[20-16])
- EX/MEM stores the control signal needed by the subsequent stages (note that execution stage control signals have already been used to control the MUXes in the EX stage.) the branch target address (output of adder), ALU output, zero flag, and destination register (output of 2:1 mux controlled by RegDst).
- MEM/WB stores control signals needed by the WB stage, ALU output and the Data memory output along with the destination register.

In this design each stage takes 1 clock cycle, meaning a single instruction will go through 5 phases and take 5 clock cycles to complete. You will still be reading an instruction in each clock cycle. Other than the registers introduced between 5 stages, the entire datapath is exactly same as the single cycle datapath.

We suggest that you complete the work by following the incremental design flow described below. First focus on controller design supporting all the operations listed in Table 1, and then implement the additional features incrementally as described below.

LAB4

Your datapath should be able to execute all instructions listed in Table 1.

TASK1 – Controller Design

You should start with the "Controller" design. A complete list of control signals required to manage your datapath is essential before attempting to implement the Controller module. Otherwise, you may risk the redesign of the entire controller.

Checkpoint 1 (during 1st week with respect to Lab4 start day)

Functionality of the following components:
ALU with your testbench
Register file with your testbench
Data memory with your testbench
Synthesis of these datapath components must be clear from primarily any latch warning

Checkpoint 2 (during 2nd week)

Functional verification of controller with your testbench
Datapath components, controller and pipelined registers are wired and top module completed

TASK2 – Datapath Design

- **Method:**
  - Design and build the datapath shown in the previous page to execute the operations listed in Table 1

Checkpoint 3 (during 3rd week)

Testbench completed
You should be executing a single instruction in behavioral simulation

Demonstration:

- Conduct post routing simulation for functional verification of each instruction
- Verify that you are able to run a sequence of all the instructions in post-routing simulation.
  - You need to create your own test program in assembly that includes all the operations
  - Translate this program to binary form and initialize your instruction/data memory with this test program.
    - You will need to learn how to initialize your instruction memory using the MIPSHelper tool (**more information below, source code given in this folder**).
      - Example:
        - We strongly recommend you to generate a program with dependent sequence of operations covering all the instructions of the ISA. An example code covering a subset of instructions is given below as a starting point. Note that the datapath figure shown in the previous page does not have any hazard detection unit. Therefore 5 "nop" instructions are needed between two instructions to make sure that dependencies are resolved before the next instruction reads from the register file. The PC is given as a reference point. It is not part of the assembly program.

```
PC=0        loop: addi $t0, $zero, $zero      # t0=0,   display 0, 0
                  nop
                  nop
                  nop
                  nop
                  nop
PC=24             addi $t1, $zero, 6          # t1= 6, display 24, 6
                  nop
                  nop
                  nop
```

```
                nop
                nop
PC=48           addi $t2, $zero, 10         # t2 = 10, display 48, 10
                nop
                nop
                nop
                nop
                nop
PC=72           sw   t1, 0($t0)        # display 72, (no register written)
                nop
                nop
                nop
                nop
                nop
PC=96           sw   t2, 4($t0)             # display 96,
                nop
                nop
                nop
                nop
                nop
PC=120          lw   s0, 0($t0)             # s0 = 6, display 120, 6
                nop
                nop
                nop
                nop
                nop
PC=144          lw   s1, 4($t0)             # s1 = 10, display 144, 10
                nop
                nop
                nop
                nop
                nop
PC=168          sub  $t3, s1, s0       # t3 = 10-6 = 4, display 168, 4
                nop
                nop
                nop
                nop
                nop
PC=192          sll  $t4, $t3, 3       # t4 = 4 << 3 = 64, display 192, 64
                nop
                nop
                nop
                nop
                nop
PC=216          srl  $t5, $t4, 2       # t5 = 16 64 >> 2, display 216, 2
                nop
                nop
                nop
                nop
                nop
PC=240          j    loop                   # display 240,
```

- Initialize your registers using “add” and addi” instructions. Note the color coded dependency between the sequential instructions above.
- Initialize your memory with a sequence of sw operations and then read from the memory with the lw.
- Display both current PC and the value written into the destination register.

- For the “sw”, “branch” and “jump” type of instructions there is no destination register therefore you should display the PC value and 0.
- It will be convenient if each operation results with a value that can be represented with at most 4 digits.
- Run the MIPShelper tool to generate your instruction and data memory modules
- Synthesize and run the program in post-routing simulation.

Demonstration:

- Functional verification by displaying the **value written into the register file after executing each instruction and the PC value of that specific instruction in post-routing simulation**.
  - **Do not change the name and port definitions for the instruction memory. During the demonstration day, we will replace your instruction memory with our version that has a precompiled program. It is highly critical that your “sw” is working properly. Our test program will write to the data memory a series of values using the “sw” instruction to initialize the contents.**
  - All you need to demo is **post-implementation functional simulation** of your data path. **Do not assume** that post-implementation will behave the same as any other simulation. If you do not have any pin assignments for your top-level outputs, Vivado will remove your entire design during implementation. During demo you will pull out two signals
    1) 32-bit program counter.
    2) 32-bit write_data to register file.

  Make sure the above 32-bit signals are retained (untrimmed) in your post routing simulations.
  Assume that first instruction of the test code corresponds to PC value of 0.

You should be using the $readmemh Verilog command when initializing your instruction and data memories. This allows you to initialize your memories from a file, with the biggest benefit being that you won’t have to resynthesize your design every time you want to change which instructions you are testing. And initializing your data memory will allow you to test your load even if your store doesn’t work.

The MIPS helper has a function to output the correct values for the instruction memory in this format, you just need to uncheck the “Output Verilog Code” checkbox, while having the “Include Original Code as Comments”, “Output 32-bit Hexadecimal Values”, and “Output Data Memory” boxes checked. This may leave some comments in what you try to copy out of it. One way you can remove the comments is to paste the output into a Google Sheet, and it should separate the hexadecimal values from the comments, where you can just copy the column with the code.

Instead of adding a text file, you should use a file with .mem file extension. When adding a text file as a source to your project (design or simulation), Vivado will not recognize the file unless you specify the absolute path. To rectify this issue, you should use a mem file instead, which uses the .mem extension. Using a mem file you will not need to specify the absolute path, and it is what we will now be using when demoing and testing.

The format for the commands is:
$readmemh("instruction_memory.mem", memory);
$readmemh("data_memory.mem", memory);
If you are having issues with readmemh, it is safer to use the absolute path to the file:
$readmemh("/home/akoglu/ece369a/memories/data_memory.mem", memory);
The file should have data structured as such, where all values are in 32-bit hexadecimal format:
00000000
00000000
00000000
00000000

Please make sure that your instruction memory can hold at least 1024 instructions. This includes using enough address bits to index into 1024 instructions.

If you have any questions, feel free to post on piazza or ask us in lab.

We will be plugging in our own “instruction_memory.mem” and “data_memory.mem” in lab on the day of the demo, using the lab computers.

- **Deliverable:**
  - Turn in only .v files used in your design (not zipped, no other format will be accepted) using designated dropbox on D2L
  - Report percentage effort in top data path module and in the testbench.
- **Penalty Conditions:**
  - Percent effort not reported (20% penalty)
  - Late submission or late demonstration (15% per day)
  - Submitting files in a folder or in compressed form (zip/tar) (10% penalty)
  - Changing the file name or extension (10% penalty)
  - Failing to demonstrate (80% penalty)
  - Design works in behavioral simulation but fails to synthesize (70% penalty)
  - Design works in behavioral simulation, synthesizes with warnings but post-routing simulation fails (60% penalty)
  - All team members must attend the demonstration
  - Unable to answer questions during the demo (up to 75% penalty and mark for competition eligibility)

- **MipsHelper** is an assembler tool implemented by Nathaniel Sema while he was taking the ece369a. Later Muneeb Mateen Ahmed, another student from ece369a fixed the known bugs and created a GUI. This latest version contained in **MipsHelper2018** folder is delivered with this assignment. This tool generates Verilog-based instruction and data memories rapidly for a given MIPS assembly code. Refer to the "MipsHelper2018" folder for the tutorial and installation instructions (**MipsHelper2018_tutorial.pdf**).

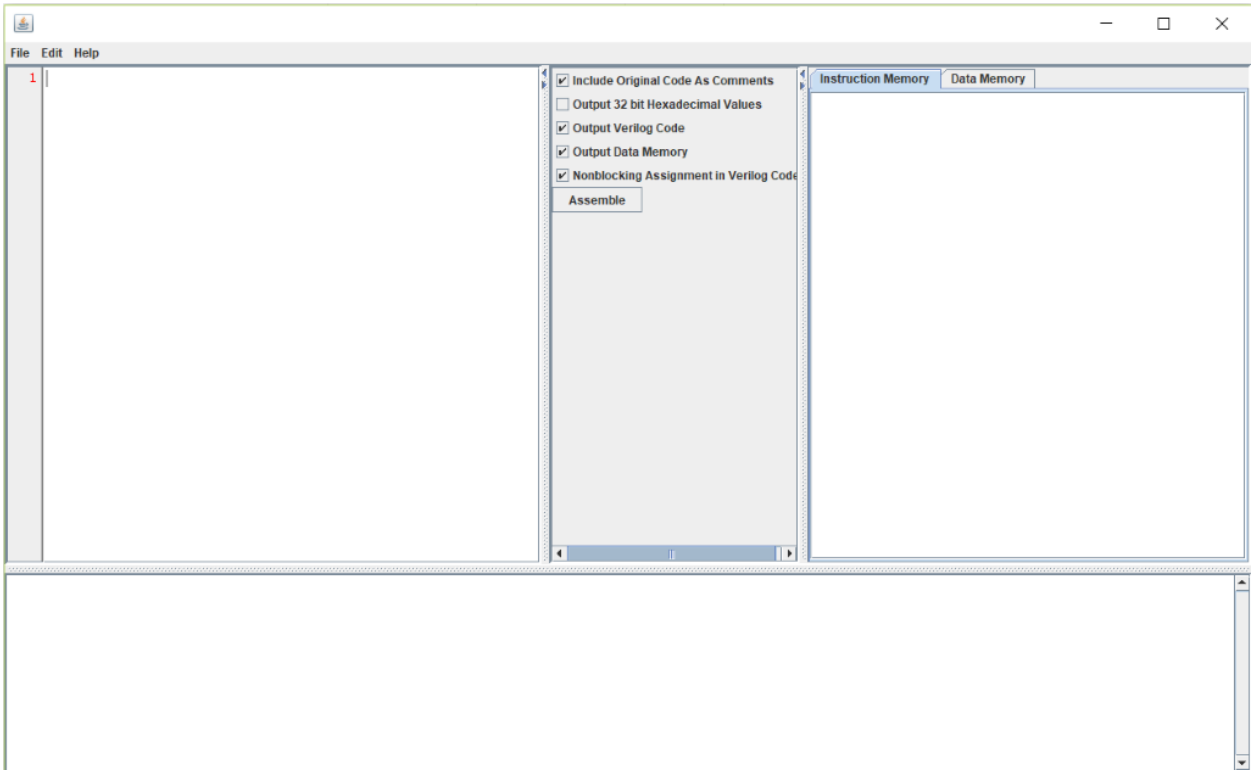


Fig. 1: Screenshot of MipsHelper2018 user interface.

Known bugs and clarifications:

1) MipsHelper2018: Every time we run the program, we get an empty file as an output. We have tried multiple formats: with $zero and $0, with commas and spaces, with tabs and spaces.

You need to use a fully functioning mips .s file with the .data, .text, and main such as:

```
.text
main:
  add $t1, $t2, $t2
  sub …
```

2) MipsHelper2018does not like hex values, you will have to convert those immediate values to base 10.

```
mipsHelper "Warning: Unknown instruction(s) found."
ori $s1, $zero, 0x0FF0
```

Lab 5

- After verifying in post-routing simulation, load your bitstream onto the FPGA
- Display the current PC value and the value that is written into the register file for that instruction on the FPGA as the datapath sequences through the instructions
- You need to use the "**Two4DigitDisplay**" and "**ClockDiv.v**" modules provided in the Lab 1 folder. This module displays two 4-digit numbers.
- We recommend testing the FPGA display with your Lab 1 solution first to make sure that you are able to display the PC on the FPGA.
- It will be convenient if each operation results with a value that can be represented with at most 4 digits (base 16).
- Note that the values that will be displayed in post-routing simulation for the test code above are base **10**. When displaying values on the FPGA, they will be **hexadecimal** form.

- **Deliverable:**
  - Turn in **.v files along with your constraint file** used in your design (not zipped, no other format will be accepted) using designated dropbox on D2L
  - Report percentage effort in top data path module and in the testbench.

- **Penalty Conditions:**
  - Percent effort not reported (20% penalty)
  - Late submission or late demonstration (15% per day)
  - Submitting files in a folder or in compressed form (zip/tar). (10% penalty)
  - Changing the file name or extension. (10% penalty)
  - Failing to demonstrate (80% penalty)
  - All team members must attend the demonstration
  - Unable to answer questions during the demo (up to 75% penalty and mark for competition eligibility)

Common Questions

Questions are organized as ISA, Synthesis and Datapath Components Related Questions.

ISA Related Questions

Q1. Overflow flag: For add, addi, sub instructions if it results in an overflow, is the destination register modified?

Answer: You don't need to worry about overflow or any other exception handling situations.

Q2. Concurrent read/write limitations - Can we read 4 registers from the register file at once. Also, can we write to register in the register file at once?

Answer: For current lab assignment we expect you to write one register at a time and read at max two registers at the same time.

Q3. SRLDoes anyone know what the 1-bit R field of SRL means. I feel like it's important as an input to the ALU.

Answer: If you compare the instruction fields for "ROTR" and "SR", they are exactly same except the one bit (Bit number 21). That R bit is used for differentiating between these two instructions. You can compare them using MIPS reference manual uploaded on D2L.
Depending on the instruction you decode in Decode stage, you will need to control the functionality of ALU for differentiating SRL and ROTR. So the additional one bit can be taken as an input in ALU control.

Q4. Unsigned/Signed differences: How can we differentiate between two similar instructions, for example add and addu? Both have the same control signals and arithmetic operations. Would we have to create a signed and unsigned version of inputs into the ALU?

Answer: Even though add and addu instructions have the same OpCode, they have different functionCode that can be used to differentiate them.

Instruction	OpCode	FunCode	ReferencePage#
add	000_000	100_000	44
addu	000_000	100_001	48

Q5. Immediate Field Issues with Test Code

```
andi   $s1 , $s1, 65535  #s1= 65535
```

My lab partner and I can't figure out how we would know that this is -1 or 65535 in the sign extender. QtSpim is able to find the difference, but both immediate fields would include all 1's. How do we set this case apart?

Answer: For "andi" a 16-bit representation of “65535" and "-1" means the same. Qtspim should respond same to -1 and 65535 for "andi".

Q6. Byte and halfword instructions
For sb,sh,lh,lb, does it automatically load/store the least significant 8 or 16 bits of the addressed word?

Answer: You would need a control for the Data Memory to select word, halfword, or byte starting at the read address
You need to add more logic to you datapath to decode and execute these instructions.
Read address will indicate which byte or half word you are going to read in a particular four-byte word.
Also, all memory access is aligned. If instruction is for half word then the instruction will point to either first half word or last half word in a particular memory word location.

Q7. sb and sh? So as far as I'm aware I have to be able to store an exact byte or halfword when data memory loads and stores words at a time. Does this mean I have to completely change how the datamemory works or am I going to have to load a full word first, put the byte/halfin and write over it? or is there a simpler solution to this that I'm not seeing?

Answer: You may need to add another control to choose word,hw,byte and appropriately mask the input to the write data of the data memory.

Q8. BGTZ: The converter converts bgtz to slt followed immediately by a bne.
memory[120] = 32'b00000000000010000000100000101010; // slt $at, $zero, $s0
memory[121] = 32'b00010100001000000000000000011000; // bne $at, $zero, branch3
Should I manually change these instructions with the BGTZ?

Answer: Yes. Change location 120 to nop and 121 to bgtz.

Q9. JAL MIPS set reference confusion: We know that JAL jumps to the designated address while also storing the address of the next instruction. However, in the MIPS set reference, it says that it should store the second instruction after JAL (PC + 8). Do we need to follow the MIPS set reference? Or should we only store PC +4 in $ra?

you store PC + 8 in the ra register instead of PC + 4 to avoid executing the subsequent instruction twice by accident if you have kept the branch delay slot in your processor.

Answer: The branch delay slot instruction, which is the instruction immediately following the jal instruction is executed before the target instruction that jal jumps to, so if you jumped back to it, then it would execute twice and that's probably not the intention of your program to do that.

If you get rid of the branch delay slot by resolving jumps and branches earlier than the datapath outline we were given, then you have to change the value that the ra register holds to PC + 4, since without the branch delay slot, you are just skipping that instruction entirely.

Otherwise, you should follow the MIPS instruction set reference.

Q10. load address? is load address the same as load word? If not on the instructions we must support but it's on the file for testcases.s. Do we need to implement the load address operation in order to properly load the DataMemory address onto $a0? When I ran the test case through mipsHelper, the first line is ori $a0, $zero, 0 rather than la $a0, DataMemory.

Answer: it's actually the same as the ORI instruction if you look at the binary. It's just loading the starting address of the instruction memory in to $a0, which in our case is 0.
You can use following:
memory[0] = 32'h34040000; // main: ori $a0, $zero, 0
instead of
la $a0, DataMemory

Synthesis and Verilog Related Questions
Q1. Outputs Must be Reg type? 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 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.

One way is to declare two temp variables for ALUResult and Zero as reg types, and then do whatever operation is needed in an always block and assign the result to those temp values. Then outside the always block assign ALUResult and Zero with whatever values are in the temp variables.

Q2. pipeline register operation: Do the pipeline registers (IF/ID, ID/EX, EX/MEM, MEM/WB) write on the rising edge of the clock and read on the falling edge of the clock, such as:

```
 always @ (posedge Clk)
    ReadData1Out <= temp;
always @ (negedge Clk)
    temp <= ReadData1In;
```

or do they just pass their inputs to their outputs like:

```
ReadData1Out <= ReadData1In;
```

Answer: Register writes should be clock edge sensitive. Depending on your design you can choose to read on a clock edge or continuously (independent of clock).

Q3. Readmem (Verilog): I want to use readmemb for instructionMemory initialization, but I don't know where to put the .txt file. My instruction memory is xxxxxx upon simulation so it seems like I have it in the wrong place.

Answer: Try adding the initialization file in your Vivado project otherwise it should be in the project "src" directory.
In Short (In Vivado):
Add Sources --> Add or Create Simulation Sources ---> Change file type to "All Files" --> Select "Instruction_memory.txt"

Q4. IM & 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 can't 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]"

Datapath Components Related Questions

Q1. For the function signExtension are we trying to preserve the signed decimal value?
Example: if 16-bit hex f000 is passed in should the output be 32-bit hex fffff000 or 32-bit hex 0000f000

Answer: Yes, this module preserves the signed bit that makes it negative.
If the number is 16'hF000 then the left most significant bit determines the sign of the number,
extended sign would make it 32'hFFFFF000 since F is 1111.

You may need both signed and unsigned versions to cover unsigned type of intrusions as well.

Q2. Opcode vs Function code for ALU Control
In the datapath diagram given, the input to ALU control is 6 least significant bits of the Instruction, but doesn't that only work for R-Type? If the Instruction is I-type, then how can the ALU still be use the correct operation? Also, some of the

instructions have the same opcode but different operations that need to happen. For example the addiu and sltiu have the same opcode (101001), but they both need to do different things in the ALU. How can that be handled?

Answer: please refer to "ALU control" slide on chapter3_single_cycle_datapath.pptx

ALU controller takes least significand 6 bits and the ALU op type to generate the control signal for the ALU. In the class I suggested processing the opcode and function fields within the big controller.

addi opcode is 001001
sltiu opcode is 001011

MIPS Reference manual gives the detailed specs

Q3. we were asked to determine the block diagram necessary to support all of the instructions listed, and the width of the ALUop signal. So my question is, why do we need to support so many MIPS instructions? Aren't we ultimately designing this processor to perform only one task? I think our assembly code only used about 10-15 different MIPS instructions.

Answer: The course and assignment objective is to expose you to MPIS instruction set, and how this instructions can be implemented in RTL. SAD implementation is just a part of it to keep this interesting. You are designing a general purpose processor. SAD is just a real-life application to test your datapath.

Q4. I am a bit confused as far as the ALU design. It says that we should be implementing all of the instructions listed, however some of the instructions (particularly branch and data instructions) seem like they wouldn't need an ALU to be completed. Should we only be focusing on the instructions that would require arithmetic operations or should our ALU take care of every single one of the operations given in the documentation?

Answer: 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.

Q5. Register file clock
Should our register file be running off of a slightly faster clock than our data path? This way it makes each phase clock change.

Answer: Your register file is a part of your datapath and all of it should operate at same clock speed. Datapath is edge sensitive, so every time there is a positive (or negative, depending on your design) the signals in your datapath will update.

Q6. Setting the ALU Zero Flag
Where is it necessary to set the zero flag in ALU? I think ALU zero flag is used for beq and bne instructions. But is it not only used in this way after a subtract operation is performed by ALU? So, could I get away with only setting the zero flag in the sub operation of ALU?

Answer: The zero flag should be set whenever the output of the ALU is zero as stated in the comments of the ALU.v

Q7. Datapath
Are we supposed to implement the exact datapath in the Word document or can we modify? The reason I ask is for the shifting instructions, we need the shamt field (bits 6 to 10) and that needs to go into the ALU but in the datapath there is no wire for that. Are we allowed to modify?

You will have to make some modifications for it to work, so yes. So long as it is still with the pipelining you should be good.

Q8. Online Datapath Maker
Has anyone found a good online datapath creator? Re-writing every time you change something is very time-consuming and messy.

Answer: Lucidchart (has and extension on google drive), or Microsoft Visio are pretty nice ways to electronically draw out the schematic. You do have to initially draw the diagram but once you have the main piece drawn out it is super easy to modify wires, or parts that you change.

You can get Visio for free using your school email.

Q9. Instruction memory: Should instruction memory be clocked to output one Instruction at every posedge of the clock?

Answer: The instruction memory should receive an input address corresponding to the desired instruction. This should come from your program counter. The program counter is clocked to update on the posedge of each clock and will provide a new address to the instruction memory. If this is set up right, there is no need for a clock in instruction memory.

Q10. Instructions repeating? I finally got my datapath to work but around pc 520 it looks like my program starts to execute the instructions from the start. Does anyone have any idea what could have caused that?

Answer: Check to make sure you changed your instruction addressing size in the module with all the instructions you generated (if it is 8:2, it is wrong); You need to change to 10:2

Q11. Synthesis Warning "Ports drive by constant Zero"
Anyone know if this will cause an issue?

Answer: Have you tried to open the Schematic? You may have a wire showing that it is hooked up and also running to ground if you have not initialized the wire in the top level design. 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.

Q12.1. Sequential element unused removing from toplevel
For some reason the implementation keeps removing a bunch of my wires because they are "unused". I made sure to initialize everything, so I have no idea why this is happening. Any ideas? I also checked and made sure that the ports being removed were connected.

Answer: During the execution of your instructions, in instruction memory, if certain logic is never being used then Vivado will trim of that logic. Since the instructions are hard coded, Vivado can narrow down on unused logic and trim it off. It should not be of concern as long as your output is correctly getting updated in post implementation simulation.

Q12.2. It is removing things like read data from register file and the Instruction as it comes out of one of the registers. So, implementation is failing because the project is empty

Answer: Your top project should have correct output and input ports. If you only have input ports and no (or incorrect) output port, then Vivado will interpret that the design is empty. Map the write port of register file on the output port of your datapath.

If you only have inputs "CLK" and "Reset" but no outputs, then your design will be empty during implementation. Pull out ALUresult as output wires in top module.

Q13. 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 doesn't in this lab. Any ideas?

Answer: Check all possibilities in case or if/else statements and make sure each output is listed there.

To avoid latches, make sure to give some default values to output and intermediate signals before entering into case or if-else statements.

Errors while trying to implement
inferring latch
referenced signal should be on sensitivity list
case item unreachable

-A latch will be inferred if some signal does not get set in all paths through your code; e.g. a switch statement that sets a variable only in most cases. We will never want a latch.
-You should probably check your sensitivity lists and make sure they cover all signals you care about- how sensitivity lists impact behavior is definitely something that changes when moving away from behavioral simulation.
-"Case item unreachable" sounds like you have defined a case outcome that never occurs, which may or may not be intentional.

Q14. Implementation Error
We were able to get every correct output in simulation, but when attempting to Implement we get this cryptic error: [Place 30-119] Unroutable DSP cascade connection found. Port 'PCOUT' of DSP block 'Hi0__1' can drive only 'PCIN' port of different DSP block.

Answer: It's probably a problem with your signed multiplication. Using $signed(A) * $signed(B) will extend A and B to 64-bit values before multiplying. (Of course, using {{32{A[31]},A} * {{32{B[31]},B}will do exactly the same thing.)

This requires a complicated routing on the FPGA, leading to your problem. I recommend only multiplying 32-bit inputs without the $signed() keyword; using $signed() will, as mentioned above, extend them to 64 bits.

To fix the problem, split the sign bits into their own 32-bit buses and work from there.

Q15. Critical warnings due to DEFAULT board connections
The Critical Warning we are getting is: Critical Warning 135 out of 135 logical ports use I/O standard (IOSTANDARD) value 'DEFAULT', instead of a user assigned specific value. We believe this is because our top-level "outputs" that we are using for waveform generation are not considered mapped to any board outputs. There are no other critical errors and our execution waveforms are exactly as we expect. Can we confirm that this type of warning will not cause any penalty in grade?

Answer: As long as you are able to see the post implementation simulations for the four signals specified in the demonstration procedure, these warnings are acceptable.

Q16. Place Design Error
I keep getting an error labeled as "place_design error". The information about it states "Check if opt_design has removed all the leaf cells of your design. Check whether you have instantiated and connected all of the top level ports." Are there some things in general that we could check to fix this issue? Google only states that this is usually a problem when there are no outputs from the top level or Reset/Clk are stuck at a value. Looking at the datapath and what we need to accomplish, I am pretty sure outputs in the top level are unnecessary, and we have tested to see if reset or clk are stuck and they are not.

Answer: on your top level, make sure to define your outputs.

Q17. [Synth 8-3332] Sequential element " " is unused and will be removed from module TopLevel.
Im getting alot of these errors and I don't understand why. I tried googling but there is no relevant solution to this case. Everything is connected the way it should be. any ideas why Im getting these messages?

Answer: To get rid of them, you can force Vivado to keep the element by putting the following statement before the declaration of a variable:

(* keep = "true" *)

so, you would declare a variable like...
(* keep = "true" *) reg [31:0] some_reg;

Q18. Post implementation wave form problem
In Vivado when i do the normal wave form my processor works just fine. But when i do implementation for some reason the synthesis "Optimizes" my design and deletes half of my processor. I'm not really sure how to fix it so that i can get my processor to run on implementation.

Answer: Go through and check for errors in the messages console. Also make sure you don't have any inferring latches.

Also, if any of the pipeline registers are not clocked, they will be treated as a wire in synthesis and implementation and therefore deleted from the design.

Q19. RegDest affecting ReadData
When we change our regDst input (to the correct input), read data no longer works, and we get XXXX, but only for the first instruction, but instructions after are fine.

Answer: Start your simulations with "Reset" active. set all the register content to 0 or some default values when reset is active. keep reset active for at least 1.50 clock cycle to be safe.

Q20.1 Pipeline IF/ID Reg to RegisterFile Timing Issue
I am having an issue with my timing between the IF/ID register and the RegisterFile itself. I don't know if this issue has been covered somewhere else, but basically when the Instruction is read from the IF/ID register on the negative edge of the clock it is then fed into the RegisterFile. When the register file gets the registers, it needs to read from it then needs to wait one more clock cycle to read the info on the negative edge of the clock and then it is fed into the ID/EX register. At this point it is a clock cycle behind from all the other data being fed into the ID/EX register. The only way I could think to fix this issue is to have the register file read asynchronously. Is there another solution to this problem that i have missed?

Answer: Your pipeline registers should "read" new data on the positive edge of the clock but always be "outputting" whatever is stored in them. This should fix your timing error.

It's also why halting the PC and the pipeline register's ability to "read" new data is important when stalling or you'll lose information you don't want to.

Q20.2 How can I have a register read on posedge of clk, but constantly output? I was having some trouble and bugs with this on my DataMemory module previously, which also implements this idea.
For example, would this work?

```
always @(posedge clk) begin
        reg1 <= input1
```

```
end
always @(*) begin
        output1<=reg1
end
```

Answer:
Simple code example:

```
input Clk, Reset;
input [31:0] Address;

output reg [31:0] Out;

always@(posedge Clk)begin
__if(Reset)begin
____Out <= 32'd0;
_end

_else begin
___Out <= Address;
_end
end
```

Now, let's say we start with clock origionally at 0 with reset high, our wave form will look like

```
Clk :     ___---___---___---___---___---___---
Reset:    ------_________________________
```

So let's break it down cycle by cycle:
Clock Cycle 0:
Wave form segment: ___
Reset is high.
Our always block is waiting for the clock to change from 0 to 1 before it does anything, so right now nothing much should be on the output signal of our code.

Posedge of Clock Cycle 1:
Wave form segment: ---
Reset is High.
Our always@ statement notices that the clock is now on the positive level and begins implementing the code found within its always block statmenet [always begin .... end].
The code takes as input the Reset value; in this case it is 1 as well as the 32-bit Address signal. However because of our if/else statements our Out Signal is 32'd0 instead of the Address' value.

Negedge of Clock Cycle 1:
Wave form segment: ___
Reset has now changed from High to Low, in all aspects the code should see that and set Out to be equal to the address, right?
Yes, but just not yet. We are still on the first clock cycle but entering the time when our clock signal is now 0 instead of 1. Because our always block is looking only at the Positive Edge of the Clock Signal it is specifically looking for when the clock signal changes from a 0 to a 1.
As such until the next clock cycle our Out signal will continuously pump-out 32'd0 on its' wires.

Posedge of Clock Cycle 2:
Wave form segment: ---

Reset is still low.
Now that the clock has changed from 0 to 1 our always block goes active again. It sees this time that Reset is 0 instead of 1. As such the Out Signal overrides whatever was on it [32'd0 in this example] and begins outputting on its' wires whatever our 32-bit Address Signal is.

It is important to realize that once you place something on an Output Wire it remains there until your override it. That is why you might see red XXXs at the beginning of your waveform but never again there-after, even if your code is no longer doing much of anything.

By saying that the register is "reading" new data on the positive edge of the clock it means that when the clock signal is changing from 0 to 1, the register is going to read whatever is on its' input wires. Based on your always block you will then determine what value gets placed onto the output wire.

Now, let us take a look at our Register Files, which are dual clocked.
On the positive edge of the clock our Register Files take in the data that is on all of the Input Wires. These then go to their respective areas within the Register File and do what they need to do.

On the negative edge of the clock our Register Files "read" the data that is within the registers and places in on the output wires (Read Data 1 and Read Data 2).
These values are constantly being outputted from the Register File until the next negative edge'd clock cycle where they are once again overwritten by some other register's values.

Essentially the instruction you want entering the Decode stage isn't arriving until the negative edge of the clock. By that time the Register File has already read from the registers, but it'll be the previous instruction's registers not your new ones. So, you've essentially opened your pipeline up to run each instruction twice.

But, to make it even more confusing your Controller is not clock based, which means that once your actual Decode instruction comes in all your Control Signals will output their appropriate commands. Unfortunately, it'll be using the incorrect data. And you'll see the correct Control Signals running through your pipeline, but not the data you should be seeing running along with them.

I'd get rid of that second always block and the parameterized register and just do always@(posedge clk) begin output1 <= input1; end

Should start to see the data and signals you were expecting to see / want to see.

Q21. Branches
My branches just won’t work, I am testing bgtz and this is my code, I tried with signed and withput the signed and it won’t work, anyone might know why? My program seems to be taking the branches all the time. I have an if statement at the end of the ALU that will make zero =1 if the ALU is 0.

```
if ($signed(A) > 0) begin
 ALUResult <= 0;
end

else begin
ALUResult <= 5;
end
```

I make aluresult 5 just so it won’t make the zero flag hi. any help thanks in advance

Answer: if statement for the zero flag is in the same always block as the operations, it doesn’t always evaluate correctly.

Try putting it in its own always block sensitive to ALU result.