**Project Deadlines and Demonstration Procedure**

There are three phases. First phase is the verification of pipelined datapath with hazard detection. Second phase is the execution of the vbsme.s on your datapath. Third phase is the competition. For clarification questions from past years that might be applicable refer to Pages 5 and beyond. Please note that some of the procedures may change to improve the fow of the demonstrations. Please check D2L for updates.

**Phase 1 – Pipelined Datapath with Hazard Detection**

* In this phase you will implement the “hazard detection unit” in the ID stage of the pipeline and execute your test code from Labs9-15 without the “nop” operations. In your revised datapath it is highly recommended that your branches and unconditional jumps are resolved in the decode stage to have a competitive advantage in cycle count category.
* The hazard detection unit should detect dependencies and eliminate the need for “nop” instructions by stalling the pipelined execution till the dependency is resolved. The hazard detection unit should check for dependencies between the current instruction in the decode stage and the instructions that are in later stages.
* Figure on the left is just an illustration of the hazard detection concept for “lw” followed by a dependent instruction case. **You will determine the complete list of inputs needed by the hazard detection unit for various dependency scenarios.** For example, the dependency on $t0 register for the following code sequence needs to be detected by the hazard detection unit to stall the pipeline.

add $t0, $t1, $t2

sub $t3, $t0, $t4

* For such dependencies, hazard detection unit needs to stall the pipeline by controlling the “PC”, “IF/ID pipeline register” and sending a “nop” instruction to the execution stage through the “MUX”.

With the Hazard Detection unit in place, your datapath now should execute the same code given in Labs9-15 document without the need of “nop” instructions as shown below:

PC=0 loop: addi $t0, $zero, $zero # t0=0, display 0, 0

PC=4 addi **$t1**, $zero, 6 # t1= 6, display 4, 6

PC=8 addi **$t2**, $zero 10 # t2 = 10, display 8, 10

PC=12 sw **t1**, 0($t0) # display 12, (no register written)

PC=16 sw **t2,** 4($t0) # display 16,

PC=20 lw **s0**, 0($t0) # s0 = 6, display 20, 6

PC=24 lw **s1**, 4($t0) # s1 = 10, display 24, 10

PC=28 sub **$t3**, **s1,** **s0** # t3 = 10-6 = 4, display 28, 4

PC=32 sll **$t4**, **$t3**, 3 # t4 = 4 << 3 = 32, display 32, 32

PC=36 srl $t5, **$t4**, 2 # t5 = 16 32 >> 2, display 36, 2

PC=40 j loop # display 40,

* + Create your own test program without “nop” instructions in assembly that includes all the instructions listed in Table 1 of Labs9-15 document.
    - Assume that first instruction of the test code corresponds to PC value of 0.
    - Translate this program to binary and initialize your instruction memory with this test program.
  + Synthesize and run the program in post-routing simulation
  + Suggested method: We strongly recommend you to generate a program with dependent sequence of operations covering all the operations.
* **Demonstration:**
  + Demo1: 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.**
    - You need to demo through post routing (implementation) simulation of your data path. During demo you will pull out four signals

1) 32 bit program counter.

2) 32 bit write\_data to register file.

3) 32 bit output of HI register.

4) 32 output of Lo register.

Make sure the above 32 bit signals are retained (untrimmed) in your post routing simulations.

* + Demo2: After verifying in post-routing simulation load your bitstream onto the FPGA
    - Verify by displaying the PC value and the value written into the register file for each instruction
    - If no value is written into register file, only display the PC value.
  + **During the demo:**
    - **TA will** **download the group submission** from d2l. Then TA will copy the instruction and data memory file into newly created project and will start the synthesis and implementation.
    - While the implementation is on-going, TA will ask questions about your implementation and details of pipelining to the team members and will take notes.
    - Once post-implementation simulation is ready, TA will check the waveform.
    - After checking waveforms TA will explain testcases that are failing if there is any. TAs won’t be sharing actual testcases.
  + **Offline Testing:**
    - Functionality of individual instructions on the pipelined datapath will be tested offline based on post-routing simulation. Your datapath should still be able to execute all the instructions from Lab9-15.
* **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
  + Include the following notes **under comment section during your submission**
    - % effort
    - Number of pipeline stages:
      * We accept both four and five stage based pipelined datapaths
      * Indicate the number of pipeline stages in your design.
    - Branch decision and resolution stage:
      * Indicate the stage (ID or EX or MEM) where you are making branch decision.
    - If you don't include any note, we will **assume** that it is a 5-stage pipeline and branches are resolved during ID stage during the offline testing.
* **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 (80% penalty)
  + Design works in behavioral simulation, synthesizes with warnings but post-routing simulation fails (70% penalty)
  + Design works in post-routing simulation but fails to run on the FPGA (30% penalty)
  + All team members must attend the demonstration (missing partner receives maximum of 50pts)
    - Unable to answer questions about your implementation during demo 30% penalty

**Phase 2**

* **Objective:**
  + Execute your vbsme on the pipelined datapath
* **Eligibility**
  + At least 75% of the instructions are functional on the datapath (Labs9-15)
  + At least 75% of the private test cases for pipelining is passing
* **Method**
  + A test case will be given to you the day before demo date. It will contain *.data* segment of vbsme private test case
  + Use the private test case to generate Instruction\_memory.mem and Data\_memory.mem with your vbsme.s with the help of MipsHelper
  + Submit the code with Instruction\_memory.mem and Data\_memory.mem. Your Instruction\_memory.v and Data\_memory.v modules should initialize memories using $readmemh and these .mem files.
  + During Demo, TA will download your project from D2L that includes all .v and .mem files along with constraint file, synthesize your design and execute the program on the **FPGA**
    - The (X,Y) coordinates of the block of the current minimum SAD should be displayed on the FPGA.
    - Display will start with (0,0) and each time a block with smaller SAD is found new coordinates should be displayed
  + Each team will then set up behavioral simulation showing
    - **PC, Current minimum, X, and Y values**
    - TA will check the simulation
  + Offline validation to be conducted by TAs
* **Deliverable**:
  + Following files are required as part of your submission:
    - All design and testbench Verilog files: "\*.v" along with constraint files
    - Data file used for initializing data and instruction memory (\*.mem)
    - .s (final form of your vbsme)
* **Penalty Conditions:**
  + Percent effort not reported (100 pt penalty)
  + Late submission (10% per day)
  + Submitting files in a folder or in compressed form (zip/tar). (20% penalty)
  + Changing the file name or extension. (20% penalty)
  + Missing any required .mem or .s file (refer to “Deliverable” item above) 50% penalty
  + Failing to demonstrate (80% penalty)
  + Design works in behavioral simulation but fails to synthesize (80% penalty)
  + Design works in behavioral simulation, synthesizes with warnings but post-routing simulation fails (70% penalty)
  + Design works in post-routing simulation, but FPGA fails to display (30% penalty)
  + All team members must attend the demonstration
    - Unable to answer questions about your implementation during demo - 50% penalty
    - Missing demonstration – 80% penalty

**Phase3:** **Competition**

* Qualification for the competition
  + No more than two non-equal workload warning
  + At least 75% of the instructions are functional on the datapath (Labs9-15)
  + Pipelined design passes at least 75% of the private test cases (Project Phase 1)
  + vbsme executes on the pipelined datapath on the FPGA (Project Phase 2)
    - you are allowed make changes to the vbsme implementation through custom instructions or change of search pattern.
  + Your datapath **must** exploit data level parallelism by invoking multiple ALU like compute units and execute vbsme in parallel.
* Discussing your ideas with the instructor is **required.** 
  + Approaches that involve exploiting instruction level parallelism through forwarding or custom instruction are not eligible for competition. Please refer to the extra credit activities listed below.
* No pipelining demo will be taken.
  + Only competition
* Your files datapath (.v) and vbsme (.s) must be uploaded on D2L (**“Phase3” folder**)
  + Your submission should have instruction memory initialized assuming the .data will have the following test case
    1. Test case will be 64x64, 4x4
* During demo, TA will download your code from D2L, generate bitstream and program the FPGA
  + TA will also see the behavioral simulation showing PC, Current minimum, X and Y values.
    1. From the behavioral simulation waveform
       - TA will collect cycle count data (**A**): number of cycles it takes to run the program
    2. From the post-routing data:
       - TA will collect Critical path delay(**B**) , BRAM usage(**C**), LUT usage(**D**), DSP usage(**E**)
    3. Results for A,B,C,D,E will be tabulated

**Extra Credit Activities**

**Implement forwarding cases** needed for your vbsme.s code, show the number of cycles it takes your vbsme code to complete without forwarding and with forwarding.

**Implement branch prediction** scheme and show the number of cycles it takes your vbsme code to complete with and without branchprediction

**Implement custom instruction(s)**, and show the number of cycles saved before and after the custom instruction using your vbsme code.

**Any other ideas:** Discuss with the instructor for approval prior to putting the design effort.

The scale of the extra credit will depend on the complexity of the modifications. All demonstrations require validation showing correct minimum SAD and the (X,Y) coordinate for the minimum SAD.

Questions From Previous Years

Q1. If you need to use stack for your SAD routing:

Stack will be implemented in data memory. On a Reset, you can initialize  $sp to the last location in the data memory.

Q2. JAL

JAL implementation and register update

For project, while implementing JAL do not add additional write port to MIPS register file (32 register file). JAL should update register $ra in the write-back stage only.

Q3. Test Cases

Sharing testcases

I was wondering if it would be alright if each group wrote their own test cases and stick them in a huge repository (git or google docs). It would help out in trying to cover the multitude of cases that can be present for this part of the lab.

We highly encourage sharing test cases.

Q4. reading instruction\_memory.mem

I placed my .txt file for instruction memory in the simulation sources folder as suggested in another post, but I still can't seem to get vivado to find it. Anyone else having this issue?

You should put this in to design sources folder. Or try adding it into project in the vivado. You will have to follow similar steps that you use for adding RTL file.

In Vivado:

Add Sources --> Add or Create Simulation Sources ---> Change file type to "All Files" --> Select "Instruction\_memory.mem"

Also, you may have to right click on the text file and change the type of file it is to "Data" before it can be read by your instruction memory. I had that issue even after I loaded the file into the project.

Q5. MIPS Helper

Creating Instruction memory to be read by readmemh

I can't seem to get the mipshelper to output hex instructions without comments. I really don;t want to go through and delete each line, so has anyone figured out how to get it to output just hex similar to "Instruction\_memory.mem" that was given to us on d2l?

You can use following commnad:

1) Run MipsHelper.

./mipsHelper369 -aiohd input.s output.txt

2) On ECE node open the text file in "VI" editor by typing "vi output.txt" on the shell then hit enter. It will open the file in VI editor.

in VI editor just type this ":%s/ ->.\*$//g" and hit enter. This will do the required changes. Save file by typing ":wq!" and then hit enter.

FQ6. FPGA Board

Output to the Board

What exactly will be the required output to the FPGA board per cycle?

For SAD on FPGA, it should be the registers tracking the coordinates (X,y) for the current minimum SAD location during the execution.

Q7. What about when we reach the end of the program? Should we put an infinite loop with a nop inside at the end of our program or something, so that we don't keep fetching more instructions?

Implement an indefinite loop like the following:

here: j here

Q8. Clock Divider

Is a Clock divider module necessary? I saw in the provided xdc file the following line:

create\_clock -period 100.000 -name Clk -waveform {0.000 5.000} [get\_ports Clk]

would changing the values in this line in the xdc essentially function to adjust the clock speed to the required setting? If so, which values would I have to change, and which values do what?

xdc file is not the place to create the clock divider. The xdc file is used for applying design constraint. Above clock command is used by the compiler as follows:

create\_clock : it indicates the signal named as "Clk" in top module is a clock signal.

-period: With this input, compiler will try to synthesize and route design on FPGA with maximum critical path of 100ns. It indicates the maximum operating frequency for your design (1/period)Hz. For competition you will have to play with this parameter to determine the minimum critical path acceptable for your design. Making it too small will give you incorrect post-implementation behavior.

-waveform: It indicates the "Clk" duty cycle. In above case it indicates that if clock starts at 0 ns then positive edge will occur at 0ns and next negedge on clock will occur on 5ns. You can change 5ns to 50ns to indicate 50% duty cycle but it wont affect your critical path. critical path will be dependent on "-period".

You will have to write a RTL module to create a clock divider which will use above clock as a reference clock input.

Q9. Instruction Memory

instruction\_memory.mem wrong on “la” instruction

In the instruction\_memory.mem - la $s2, asize1: We know that the la instruction is the equivalent of lui. However, if you convert the hex instruction to binary and compare the opcode to what lui is supposed to be, it is different. The opcode turns out to be the opcode for an ori instruction, which causes unexpected results. Is this an error with the MIPSHelper?

In qtspim, we can not access data memory at address location 0 as initial address space is reserved for system specific code so we need to use "la". In your HDL implementation, memory location 0 is accessible to you. So mipsHelper will take the starting memory address as location 0 and will convert "la" to "ori" to load correct memory address.

Q10 could not open $readmem data file "Instruction\_memory.mem"

[Synth 8-4445] could not open $readmem data file 'Instruction\_memory.mem'; please make sure the file is added to project and has read permission, ignoring ["C:/Users/netid/Downloads/InstrMem.v":923]

I'm getting this error after implementation, although the Instruction\_memory.mem is in simulation sources and is a memory initilization file. Does anyone know the fix?

It has to be added as a design source as well for it to work for synthesizing and implementation.

EDIT: you also might have to right click on the memory files after they're added to the project and set the type as memory initialization files.

Q11. Data Memory

Phase 1 Data Memory

The project requirements state we only need to read from the .mem file for the instruction memory. What should data memory be initialized to for the set of private test cases we are getting for the phase 1 test? Do we leave the data memory as it is from our own tests, Set it to all zeroes, or does it also need to read from some data\_memory.mem etc. ?

You will be reading data memory from a memory file. Name it as "data\_memory.mem".

When creating your own testcases .s file, the data memory may come from the lines at the top of the file before '.text'. You can use MipsHelper to convert those lines into a datamemory.mem file by running:

./mipsHelper -aiodh ./testcases.s ./out.txt

This will produce two files, 'out.txt' and 'out\_data.txt'. Since the instruction memory is already given to us in the proper format, you can ignore the first. The second will essentially be a Verilog file that you could copy into the data memory module. However, in order to make it readable, (it should be fairly short anyways) you should take each line and replace 'memory[n] = 32'h100' (for example) with the plain hexadecimal number, 00000064.

Q12. initialize Data memory

Does anyone know what value to initialize the data memory in order to get the list to load?

.data

asize0: .word 100, 200, 300, 400, 500, 600

asize1: .word 700, 800, 900, 1000, 1100, 1200

.text

If you are looking to create the file for data Memory you can use the mipsHelper with the '-d' command and it will create a data memory file in addition to the instruction memory file, for the demo you will need to read from a file so I recommend that way. If you want to hard code a few values the memory of asize0 starts with position 0, and increments up from there. So it would look like: memory[0] = 100; memory[1] = 200; etc and asize1 starts at position 7.

Q13. Timing Analysis

I am currently trying to run the timing analysis of our datapath and I am following the outline provided in the Verilog folder in D2L and it is showing a Clock in step 8). Is that clock from the Clock Divider in our project? I don't seem to have that in my timing analysis which makes me believe that I currently don't have an external clock running my project.

You can put same timing constraints on both the clock, input clock to your wrapper and output of clock divider (input to your datapath). But only input clock to wrapper should be assigned a pin number on the FPGA.

Q14. Issue with $readmemh and Instruction Memory

We're trying to read Instruction\_memory.mem into our InstructionMemory module, but we keep running into this error, no matter how we add the text file to our project.

"could not open $readmem data file 'Instruction\_memory.mem"; please make sure the file is added to project and has read permission"

We've already added instruction\_memory.mem to the simulation sources, but the same error still shows. Has anyone else run into this problem?

download file> add sources>add design source>(find the file)> all files(dropdown)> finish>(at this point the file should be in a folder in your project called text files)> open that folder> right click file> setfile type> memory initialization file

Q15. Question on $readmemh

The notation given to us was "$readmemh ("Instruction\_memory.mem", <memory register identifiers>)" and some lines of code in the text file are:

3c120000

8e520000

3c130000

8e730004

02538820

Which are hexadecimal values. Does that mean $readmemh ONLY accepts hex values? I've been trying to put my own binary values such as:

00100000000100000000000000000001

00000000000000000000000000000000

00000000000000000000000000000000

00000000000000000000000000000000

00000000000000000000000000000000

00000000000000000000000000000000

00100000000100010000000000000001

to be read but it isn't processing correctly. Do I simply need to convert these binary values to hex values for it to be read by $readmemh correctly? I would think $readmemh would be able to read the value in the text value no matter what notation it's in.

readmemh: to read hexadecimal

Also, you can use the h flag in mipshelper to have the output be in hex.

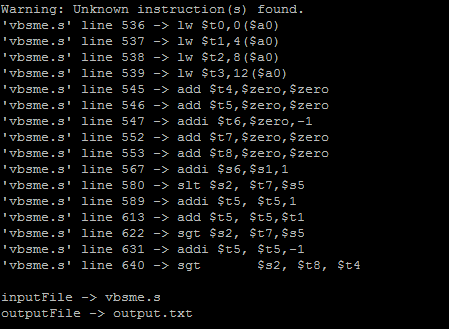
Q16. test case wrong?

the ori messes up my execution. I've come to the conclusion that I've either gone crazy, or something is wrong in the file. The ORI instruction uses 0xaaaa as the immediate field, and I searched for this value in hex, and decimal, and couldn't find it in my entire datapath... it doesn't exist. The instruction is correct in my memory. Anyone else run into this issue?

Andi ori and xori all are zero extended and not sign extended.

Q17. Mips helper giving errors on LW

When trying to convert our VBSME I am getting unknown instructions on known instructions. looking for advice.



you need to insert space in between registers for each instruction.

for example instead of : lw $t0,0($a0) it should be: lw $t0, 0($a0). pay attention to space between $t0 and 0($a0),