CprE 381: Computer Organization and Assembly-Level Programming

Project Part 2 Report

Team Members: Cristofer Medina Lopez

Patrick Bruce

Project Teams Group #: Section 3_2

Refer to the highlighted language in the project 1 instruction for the context of the following questions.

[1.a] Come up with a global list of the datapath values and control signals that are required during each pipeline stage.

Fetch-Decode :

- Instruction (instr, 32-bits)
- Program Counter Address (pc_val, 32-bit)

Decode-Execute:

- RegDst
- RegWrite
- ALUOp
- ALUSrc
- Branch
- MemWrite

- MemtoReg
- Jump
- BNE
- JAL
- JR
- Instruction (instr, 32-bits)
- Register Read Output 1 (regout0, 32-bits)
- Register Read Output 2 (regout1, 32-bits)
- Immediate Value (immi, 32-bits)
- Program Counter Address (pc_val, 32-bits)

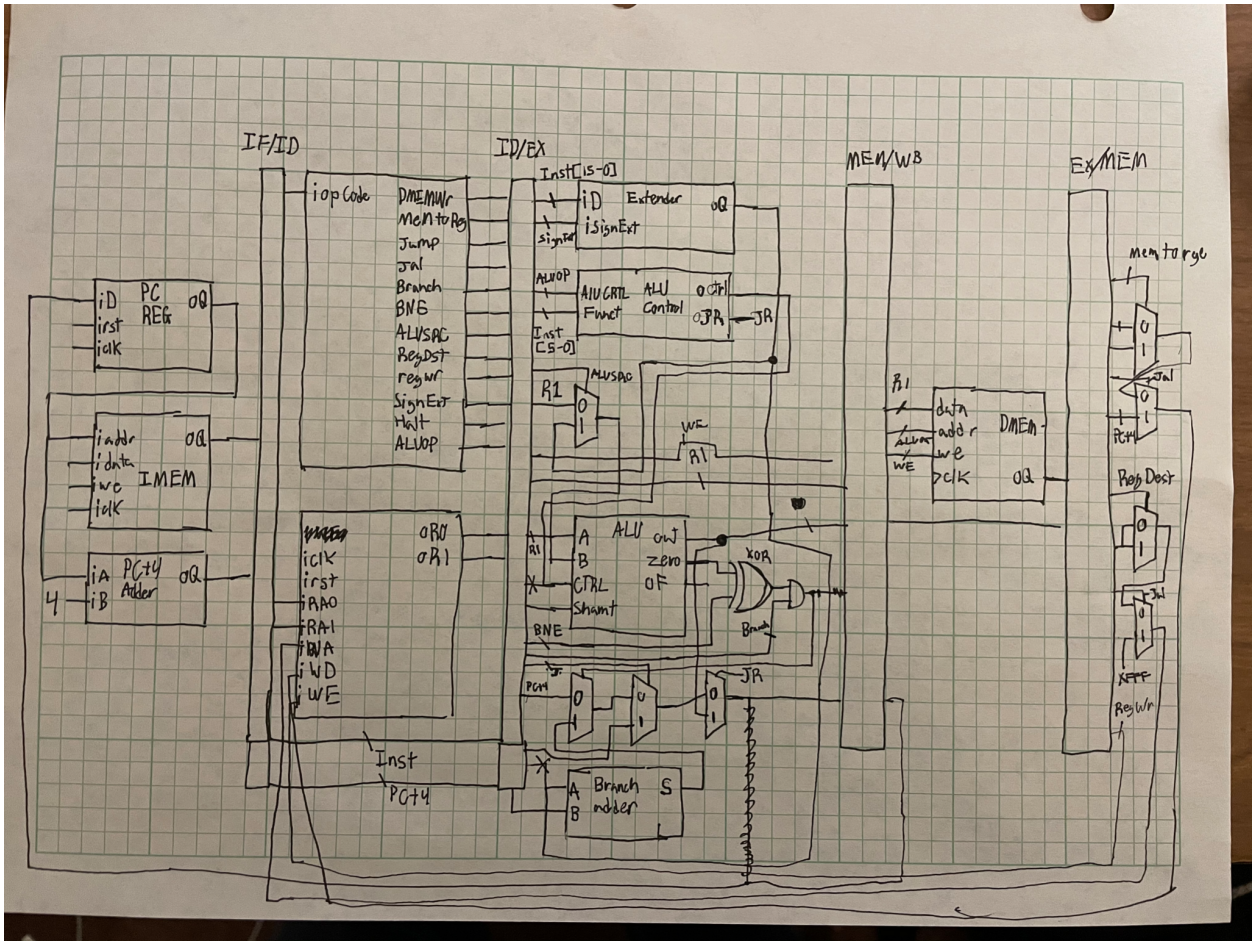
Execute-Memory:

- RegDst
- RegWrite
- Branch
- MemWrite
- MemtoReg
- Jump
- BNE
- JAL
- JR
- Instruction (instr, 32-bits)
- Register Write Address (regDst, 5-bits)
- Register Read Output 2 (regout1, 32-bits)
- ALU Result (AluResult, 32-bit)
- Program Counter Address (pc_val, 32-bits)

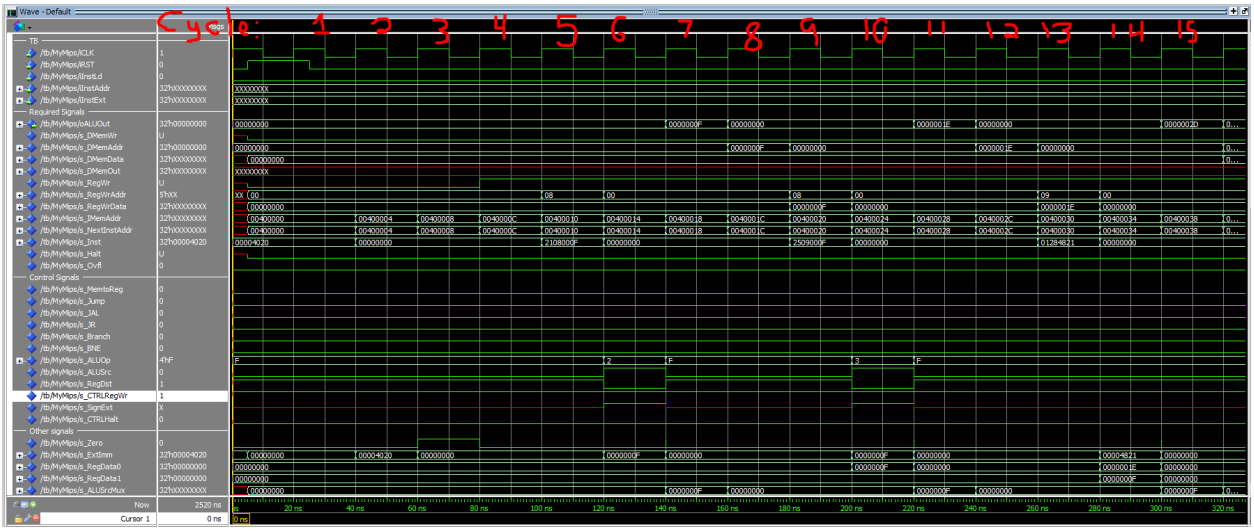
Memory-Write Back:

- RegWrite
- Branch
- Jump
- BNE
- JAL
- JR
- MemtoReg
- Instruction (instr, 32-bits)
- Memory Out Data (MemData, 32-bits)
- ALU Result (AluResult, 32-bit)
- Program Counter Address (pc_val, 32-bits)

[1.b.ii] high-level schematic drawing of the interconnection between components.



[1.c.i] include an annotated waveform in your writeup and provide a short discussion of result correctness.



The test was successful as it ran both on mars and on the test bench using our processor. The reason for the success on our processor is the use of nops to avoid hazards. The first few instructions use the same registers. It begins with an add on cycle one which stores into register 8. 3 nops need to be inserted and then the next operation can occur. On cycle 5 an add is performed using register 8 as a source this time.

The full wave form is included in a file titled “Proj2_base_test.wlf”.

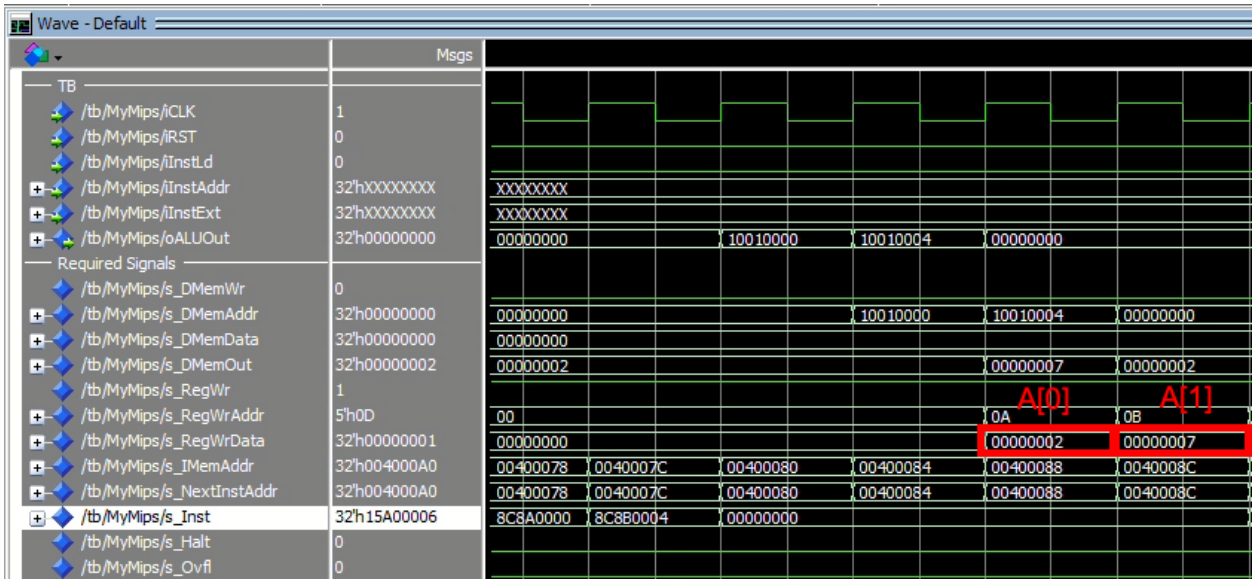
[1.c.ii] Include an annotated waveform in your writeup of two iterations or recursions of these programs executing correctly and provide a short discussion of result correctness. In your waveform and annotation, provide 3 different examples (at least one data-flow and one control-flow) of where you did not have to use the maximum number of NOPs.

The bubble sort algorithm from the signal cycle design was modified through the addition of NOPs to avoid hazards. This ran successfully on both Mars and the pipeline processor.

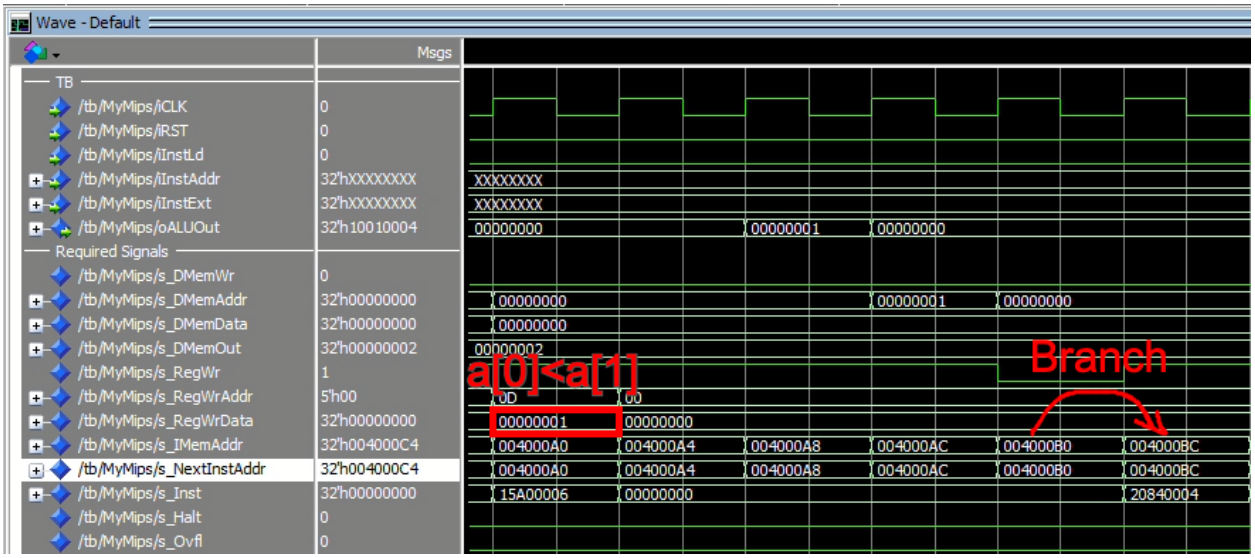
NOPs were added only when necessary and rescheduling was used to avoid additional NOPs.

Two iterations of bubble sort working correctly

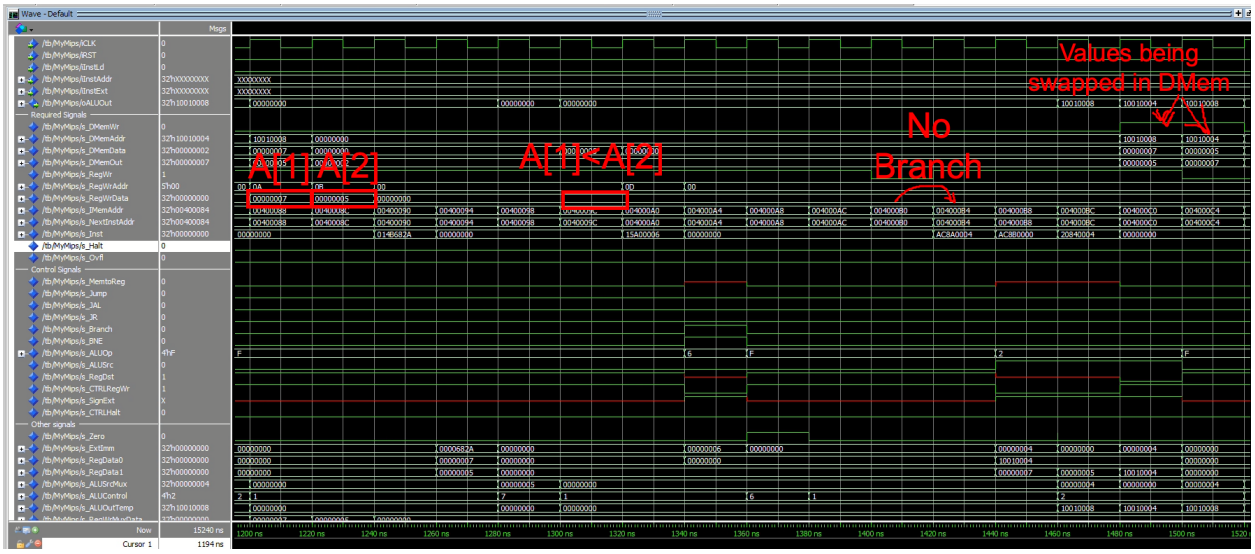
This first image shows the start of the meat of the program. 2 array values are loaded from memory and stored into the register to be compared. The pipeline can be observed in action as the RegWrData signals are asserted to the values of DMemOut from the previous cycle.



In the next image the comparison between the 2 values occurs. A[0] is smaller than A[1] so a branch occurs that skips the swap operation and moves on to the next pair of elements showing the success of bubble sort on this iteration.



The next iteration is the same except this time the comparison is between A[1] = 7 and A[2] = 5. In this case a swap is needed and can be seen in the waveform below. This further confirms the correctness of the algorithm.



3 Instances of Saving Nops

The waveform and code below shows an instance where rescheduling saves a nop from being exicuted. In the original code the “addi” and “add” instructions were swapped. Swapping them to the new sequence allows for one less nop because of “beq’s” dependance on “addi”

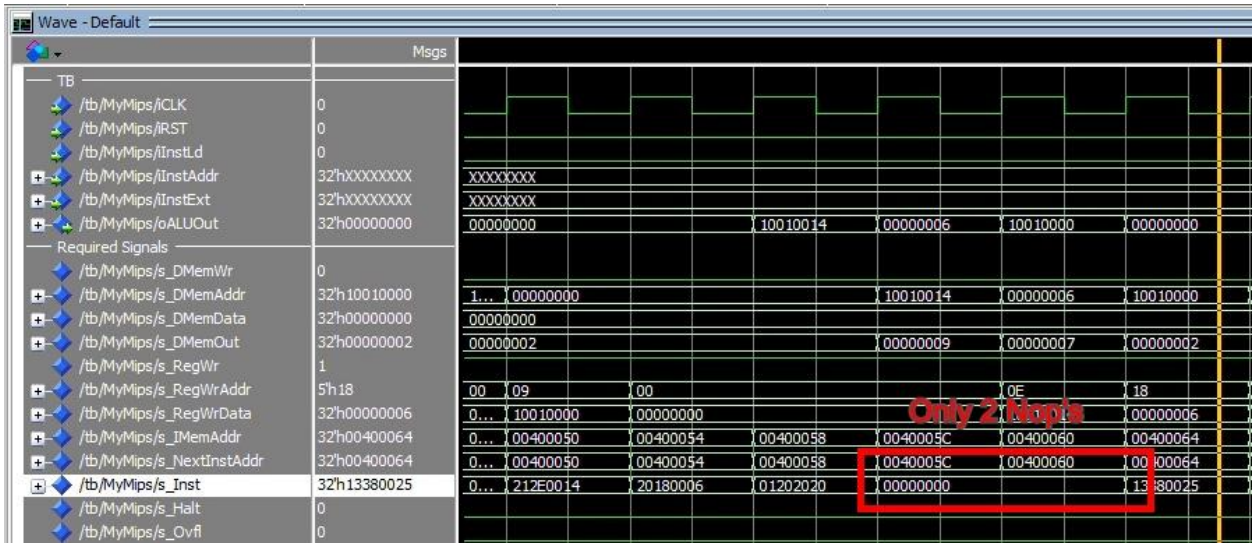
```
addi $t8, $0, 6          # size = 6 elements

add $a0, $t1, $0         # outer loop will reset inner loop
```

nop

nop

beq $t9, $t8, end # iterations = size; leave loop - array is sorted



The next wave form and code shows another example where rescheduling is able to save a no op due to a data hazard.

Copy the Array address

lui $a0, 0x1001

sort loop iterator(outer loop) i = 1

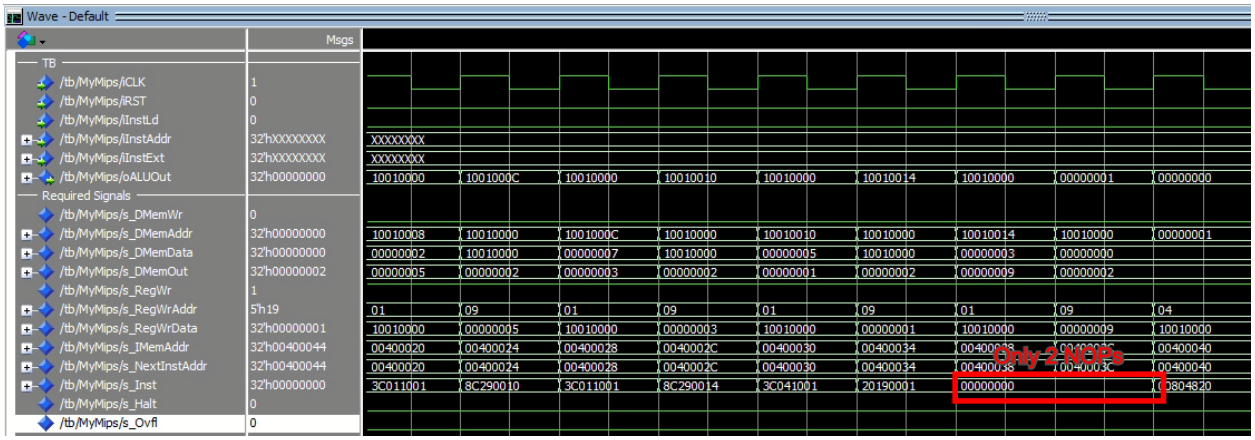
addi $t9, $0, 1

nop

nop

Base address of array

add $t1, $a0, $0



The final example takes advantage of the delay slots. We cannot show a wave form of this case as it will not run on MARs. This is beauce mars does not have delay slots to utilize. However, the code below shows that by putting the code after the jump, the nop that would have been run there to avoid control hazards is now a useful instruction. This is possible because the jump does not depend on that instruction at all.

```
j outer_loop

addi $t9, $t9, 1                # i++

Nop
```

For more information, the file “Proj2_bubblesort.wlf” contains the full waveform and is included with this submission

[1.d] report the maximum frequency your software-scheduled pipelined processor can run at and determine what your critical path is (specify each module/entity/component that this path goes through).

The maximum frequency of the processor is 66.08 MHz

The critical path takes the following route:

ID/EX pipeline -> ALU data mux -> ALU -> EX/MEM Pipeline