CprE 381: Computer Organization and Assembly-Level Programming

Project Part 1 Report

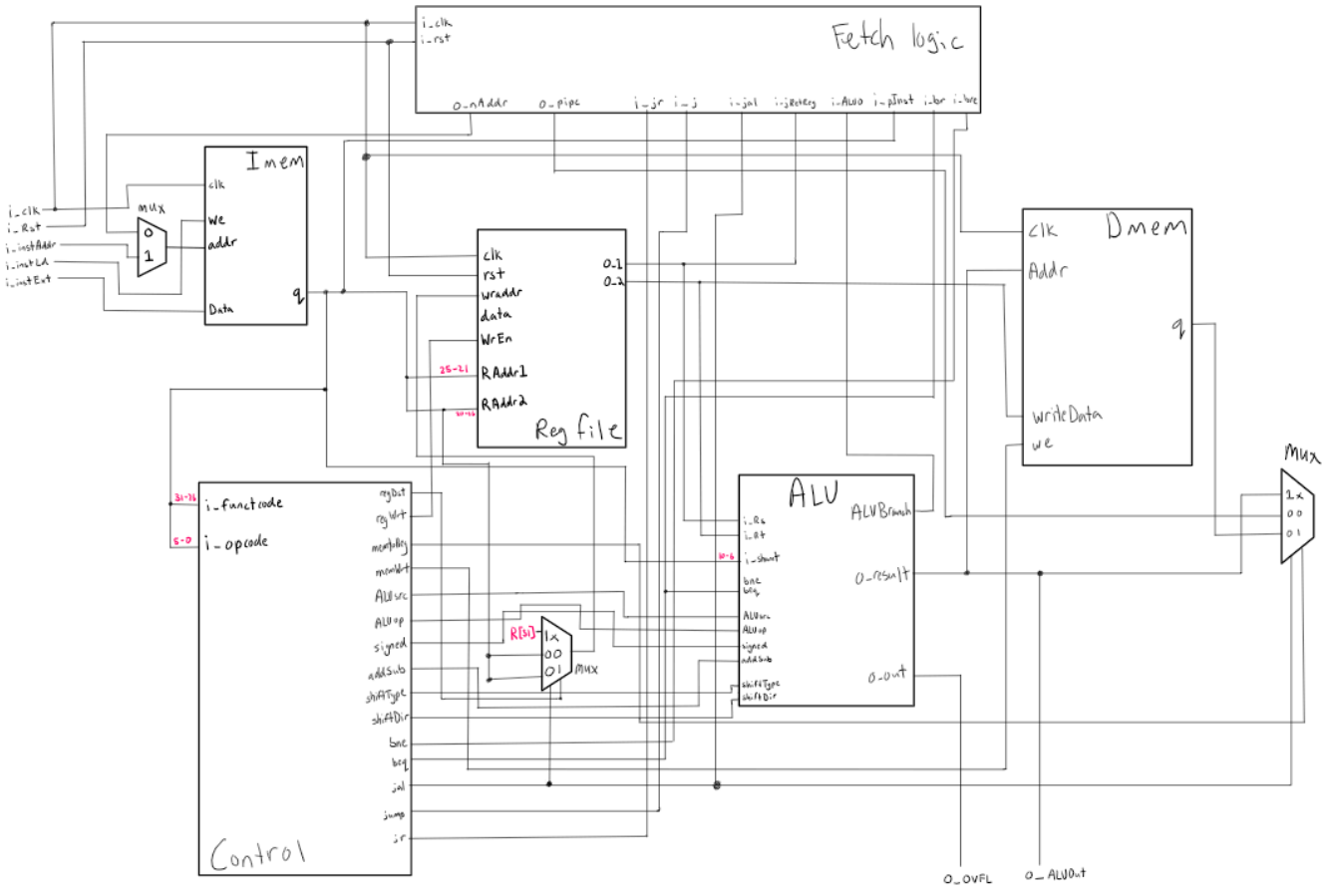
Team Members: Abrahim Toutoungi

Mitchell Driscoll

Project Teams Group #: 06

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

[Part 1 (d)] Include your final MIPS processor schematic in your lab report.



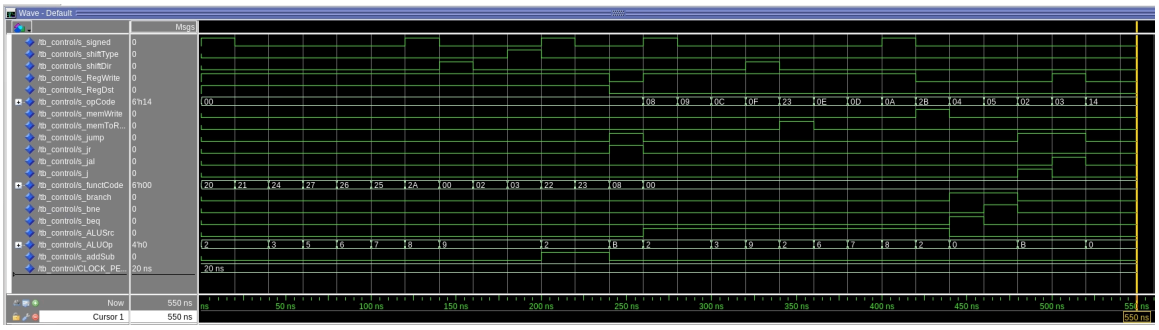
[Part 2 (a.i)] Create a spreadsheet detailing the list of *M* instructions to be supported in your project alongside their binary opcodes and funct fields, if applicable. Create a

separate column for each binary bit. Inside this spreadsheet, create a new column for the *N* control signals needed by your datapath implementation. The end result should be an *N*M* table where each row corresponds to the output of the control logic module for a given instruction.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Instruction	Opcode (Binary)	Funct (Binary)											
2				ALUSrc	ALUOp	ALUControl	MemtoReg	MemRead	MemWr	RegWr	RegDst	Jump	Branch	Type
3	add	"000000"	"100000"	0	"10"	"0010"	0	0	0	1	1 [rd]	0	0	R
4	addu	"000000"	"100001"	0	"10"	"0010"	0	0	0	1	1	0	0	R
5	and	"000000"	"100100"	0	"10"	"0000"	0	0	0	1	1	0	0	R
6	nor	"000000"	"100111"	0	"10"	"0101"	0	0	0	1	1	0	0	R
7	xor	"000000"	"100110"	0	"10"	"0011"	0	0	0	1	1	0	0	R
8	or	"000000"	"100101"	0	"10"	"0001"	0	0	0	1	1	0	0	R
9	slt	"000000"	"101010"	0	"10"	"0111"	0	0	0	1	1	0	0	R
10	sll	"000000"	"000000"	0	"10"	"0100"	0	0	0	1	1	0	0	R
11	srl	"000000"	"000010"	0	"10"	"0100"	0	0	0	1	1	0	0	R
12	sra	"000000"	"000011"	0	"10"	"0100"	0	0	0	1	1	0	0	R
13	sub	"000000"	"100010"	0	"10"	"0110"	0	0	0	1	1	0	0	R
14	subu	"000000"	"100011"	0	"10"	"0110"	0	0	0	1	1	0	0	R
15	jr	"000000"	"001000"	x/0	"10"	"0010"	0	0	0	1	x/1	0	0	R
16	sllv	"000000"	"000100"	0	"10"	"0100"	0	0	0	1	x/1	0	0	R
17	srlv	"000000"	"000110"	0	"10"	"0100"	0	0	0	1	x/1	0	0	R
18	srav	"000000"	"000111"	0	"10"	"0100"	0	0	0	1	x/1	0	0	R
19	beq	"000100"	"------"	x/1	"01"	"0110"	x/0	0	0	0	x	0	1	I
20	bne	"000101	"------"	x/1	"01"	"0110"	0	0	0	0	x	0	1	I
21	addi	"001000"	"------"	1	"00"	"0010"	0	0	0	1	0 [rt]	0	0	I
22	addiu	"001001"	"------"	1	"00"	"0010"	0	0	0	1	0	0	0	I
23	slti	"001010"	"------"	1	"00"	"0111"	0	0	0	1	0	0	0	I

Attached .xls in zip submission.

[Part 2 (a.ii)] Implement the control logic module using whatever method and coding style you prefer. Create a testbench to test this module individually and show that your output matches the expected control signals from problem 1(a).

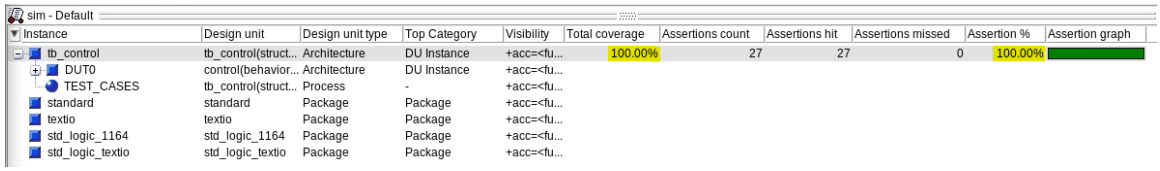


The above waveforms demonstrate that the control logic is working correctly. Since the waveforms are a little trickier to follow, we used asserts, which we'll discuss below. In the waveform diagram above, we can see that the first 13 tests are R-type instructions with all 0 opcodes. The ALUOp output was correct for all of them.

We came across asserts in VHDL, and they have made writing and checking tests a lot easier. Here is an example of a test with an assert:

```
98      s_opCode <= "000000";
99      s_functCode <= "100000";
100     WAIT FOR CLOCK_PERIOD; --add instruction
101     ASSERT s_ALUOp = b"0010" REPORT "Wrong ALUOp" SEVERITY error;
```

We checked to ensure the ALUOp output was correct because that is what changes based on the opcode and functcode. We have a total of 27 tests, and each of them is written like the one above.



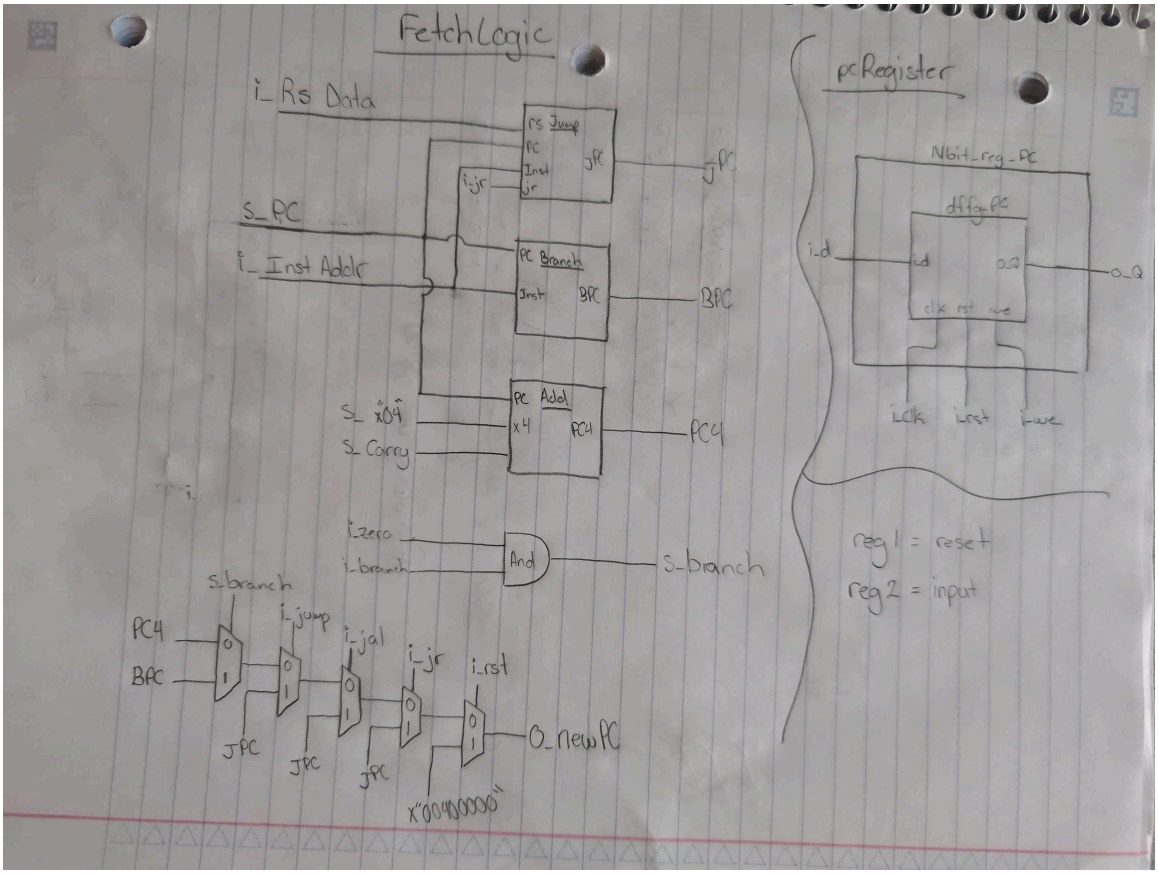
This picture shows that all the assertions hit.

[Part 2 (b.i)] What are the control flow possibilities that your instruction fetch logic must support? Describe these possibilities as a function of the different control flow-related instructions you are required to implement.

The instruction fetch logic must support various control flow possibilities based on the different control flow-related instructions required to implement. These possibilities can be described as follows:

1. **Non-Control Flow:** In this scenario, the program counter (PC) is incremented by 4, indicating sequential execution of instructions.
2. **Jump (j):** The PC is set to the value specified by JumpAddr, which is calculated as { PC+4[31:28], address, 2'b0 }.
3. **Jump Register (jr):** The PC is set to the value stored in register R[rs].
4. **Jump and Link (jal):** This instruction involves two steps:
   R[31] is set to PC + 8.
   PC is set to the value specified by JumpAddr.
5. **Branch on Equal (beq):** If the contents of registers R[rs] and R[rt] are equal, the PC is updated using the formula PC = PC + 4 + BranchAddr.
6. **Branch Not Equal (bne):** If the contents of registers R[rs] and R[rt] are not equal, the PC is updated using the formula PC = PC + 4 + BranchAddr.

[Part 2 (b.ii)] Draw a schematic for the instruction fetch logic and any other datapath modifications needed for control flow instructions. What additional control signals are needed?



[Part 2 (b.iii)] Implement your new instruction fetch logic using VHDL. Use QuestaSim to test your design thoroughly to make sure it is working as expected. Describe how the execution of the control flow possibilities corresponds to the QuestaSim waveforms in your writeup.

[Part 2 (c.i.1)] Describe the difference between logical (`srl`) and arithmetic (`sra`) shifts. Why does MIPS not have a `sla` instruction?

The distinction between logical (SRL) and arithmetic (SRA) shifts lies in how they handle shifting operations, particularly with signed values.

Shift Right Logical (SRL): This operation shifts the bits of a value to the right, filling the vacant leftmost bits with zeros. While this is straightforward and efficient for unsigned values, it can lead to unexpected results with signed values. This is because the insertion of zeros doesn't consider the sign of the value, potentially altering the sign bit and causing unexpected behavior.

Shift Right Arithmetic (SRA): Unlike SRL, SRA takes into account the sign of the value being shifted. It shifts the bits to the right but fills the vacant leftmost bits with

copies of the sign bit, maintaining the sign and ensuring predictable results, especially with signed values.

MIPS doesn't have a SLA (Shift Left Arithmetic) instruction because SLA could lead to unpredictable outcomes, particularly with signed values. SLA would insert the sign bit into the least significant bit position when shifting left. However, since the most significant bit would eventually be shifted out, the result could flip from positive to negative or vice versa, depending on the current MSB and the number of shifts. This unpredictability makes SLA less useful and potentially problematic for maintaining the integrity of signed values during shifting operations. Therefore, MIPS architecture opts not to include an SLA instruction to avoid such issues.

[Part 2 (c.i.2)] In your writeup, briefly describe how your VHDL code implements both the arithmetic and logical shifting operations.

[Part 2 (c.i.3)] In your writeup, explain how the right barrel shifter above can be enhanced to also support left shifting operations.

[Part 2 (c.i.4)] Describe how the execution of the different shifting operations corresponds to the QuestaSim waveforms in your writeup.

[Part 2 (c.ii.1)] In your writeup, briefly describe your design approach, including any resources you used to choose or implement the design. Include at least one design decision you had to make.

[Part 2 (c.ii.2)] Describe how the execution of the different operations corresponds to the QuestaSim waveforms in your writeup.

[Part 2 (c.iii)] Draw a simplified, high-level schematic for the 32-bit ALU. Consider the following questions: how is Overflow calculated? How is Zero calculated? How is `slt` implemented?

[Part 2 (c.v)] Describe how the execution of the different operations corresponds to the QuestaSim waveforms in your writeup.

[Part 2 (c.viii)] justify why your test plan is comprehensive. **Include waveforms** that demonstrate your test programs functioning.

[Part 3] In your writeup, show the QuestaSim output for each of the following tests, and provide a discussion of result correctness. It may be helpful to also annotate the waveforms directly.

[Part 3 (a)] Create a test application that makes use of every required arithmetic/logical instruction at least once. The application need not perform any particularly useful task, but it should demonstrate the full functionality of the processor (e.g., sequences of many instructions executed sequentially, 1 per cycle while data written into registers can be effectively retrieved and used by later instructions). Name this file Proj1_base_test.s.

[Part 3 (b)] Create and test an application which uses each of the required control-flow instructions and has a call depth of at least 5 (i.e., the number of activation records on the stack is at least 4). Name this file Proj1_cf_test.s.

[Part 3 (c)] Create and test an application that sorts an array with *N* elements using the BubbleSort algorithm (link). Name this file Proj1_bubblesort.s.

[Part 4] Report the maximum frequency your processor can run at and determine what your critical path is. Draw this critical path on top of your top-level schematic from part 1. What components would you focus on to improve the frequency?

FMax: 22.54mhz Clk Constraint: 20.00ns Slack: -24.37ns

The path is given below

```
 ====================================================================
 From Node    : pcRegister:PC_REG|Nbit_reg_PC:REG|dffg_PC:\G_NBit_Reg:9:REG|s_Q
 To Node      : MIPSregister:G_REG|Nbit_reg:\regi:21:REG|dffg:\G_NBit_Reg:20:REG|s_Q
 Launch Clock : iCLK
 Latch Clock  : iCLK
 Data Arrival Path:
 Total (ns)  Incr (ns)     Type  Element
 ==========  ========= ==  ====  ===================================
      0.000     0.000           launch edge time
      3.091     3.091  R        clock network delay
      3.323     0.232     uTco  pcRegister:PC_REG|Nbit_reg_PC:REG|dffg_PC:\G_NBit_Reg:9:REG|s_Q
      3.323     0.000 FF  CELL  PC_REG|REG|\G_NBit_Reg:9:REG|s_Q|q
      4.692     1.369 FF    IC  s_IMemAddr[9]~4|datad
      4.817     0.125 FF  CELL  s_IMemAddr[9]~4|combout
      7.815     2.998 FF    IC  IMem|ram~45332|dataa
      8.239     0.424 FF  CELL  IMem|ram~45332|combout
      8.506     0.267 FF    IC  IMem|ram~45333|datab
      8.931     0.425 FF  CELL  IMem|ram~45333|combout
     10.628     1.697 FF    IC  IMem|ram~45336|datad
     10.753     0.125 FF  CELL  IMem|ram~45336|combout
     10.980     0.227 FF    IC  IMem|ram~45339|datad
     11.105     0.125 FF  CELL  IMem|ram~45339|combout
     11.332     0.227 FF    IC  IMem|ram~45340|datad
     11.457     0.125 FF  CELL  IMem|ram~45340|combout
     13.219     1.762 FF    IC  IMem|ram~45383|datac
     13.500     0.281 FF  CELL  IMem|ram~45383|combout
     13.767     0.267 FF    IC  IMem|ram~45426|datab
     14.171     0.404 FF  CELL  IMem|ram~45426|combout
     14.406     0.235 FF    IC  IMem|ram~45597|datac
     14.687     0.281 FF  CELL  IMem|ram~45597|combout
     14.912     0.225 FF    IC  IMem|ram~45768|datad
     15.062     0.150 FR  CELL  IMem|ram~45768|combout
     15.753     0.691 RR    IC  G_REG|mux2|mx_out[10]~20|datad
     15.908     0.155 RR  CELL  G_REG|mux2|mx_out[10]~20|combout
     16.680     0.772 RR    IC  G_REG|mux2|mx_out[2]~548|datad
     16.835     0.155 RR  CELL  G_REG|mux2|mx_out[2]~548|combout
     19.671     2.836 RR    IC  G_REG|mux2|mx_out[2]~549|datad
     19.826     0.155 RR  CELL  G_REG|mux2|mx_out[2]~549|combout
     20.033     0.207 RR    IC  G_REG|mux2|mx_out[2]~553|datad
     20.188     0.155 RR  CELL  G_REG|mux2|mx_out[2]~553|combout
     20.390     0.202 RR    IC  G_REG|mux2|mx_out[2]~556|datad
     20.545     0.155 RR  CELL  G_REG|mux2|mx_out[2]~556|combout
     21.514     0.969 RR    IC  G_ALU|G_MUX_IMM|\G_NBit_MUX:2:MUXI|o_O~0|datad
     21.669     0.155 RR  CELL  G_ALU|G_MUX_IMM|\G_NBit_MUX:2:MUXI|o_O~0|combout
```

```
    21.669      0.155 RR  CELL  G_ALU|G_MUX_IMM|\G_NBit_MUX:2:MUXI|o_O~0|combout
    22.129      0.460 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:2:ADDI1|OR1|o_F~0|datad
    22.284      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:2:ADDI1|OR1|o_F~0|combout
    22.512      0.228 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:3:ADDI1|OR1|o_F~0|datad
    22.667      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:3:ADDI1|OR1|o_F~0|combout
    22.894      0.227 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:4:ADDI1|OR1|o_F~0|datad
    23.049      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:4:ADDI1|OR1|o_F~0|combout
    23.277      0.228 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:5:ADDI1|OR1|o_F~0|datad
    23.432      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:5:ADDI1|OR1|o_F~0|combout
    23.660      0.228 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:6:ADDI1|OR1|o_F~0|datad
    23.815      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:6:ADDI1|OR1|o_F~0|combout
    24.029      0.214 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:7:ADDI1|OR1|o_F~0|datad
    24.184      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:7:ADDI1|OR1|o_F~0|combout
    24.881      0.697 RR    IC  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:8:ADDI1|XOR2|o_F|datad
    25.036      0.155 RR  CELL  G_ALU|G_ADDSUB|N_Bit_Adder|\G_NBit_Adder:8:ADDI1|XOR2|o_F|combout
    25.240      0.204 RR    IC  G_ALU|G_SELECT|o_result[8]~116|datad
    25.395      0.155 RR  CELL  G_ALU|G_SELECT|o_result[8]~116|combout
    25.598      0.203 RR    IC  G_ALU|G_SELECT|o_result[8]~117|datad
    25.753      0.155 RR  CELL  G_ALU|G_SELECT|o_result[8]~117|combout
    26.440      0.687 RR    IC  G_ALU|G_SELECT|o_result[8]~119|datac
    26.727      0.287 RR  CELL  G_ALU|G_SELECT|o_result[8]~119|combout
    26.930      0.203 RR    IC  G_ALU|G_SELECT|o_result[8]~230|datad
    27.085      0.155 RR  CELL  G_ALU|G_SELECT|o_result[8]~230|combout
    29.262      2.177 RR    IC  DMem|ram~37013|dataa
    29.659      0.397 RR  CELL  DMem|ram~37013|combout
    40.492     10.833 RR    IC  DMem|ram~37014|datac
    40.779      0.287 RR  CELL  DMem|ram~37014|combout
    40.983      0.204 RR    IC  DMem|ram~37015|datad
    41.138      0.155 RR  CELL  DMem|ram~37015|combout
    41.341      0.203 RR    IC  DMem|ram~37026|datad
    41.496      0.155 RR  CELL  DMem|ram~37026|combout
    42.209      0.713 RR    IC  DMem|ram~37027|datac
    42.496      0.287 RR  CELL  DMem|ram~37027|combout
    42.703      0.207 RR    IC  DMem|ram~37070|datad
    42.842      0.139 RF  CELL  DMem|ram~37070|combout
    43.117      0.275 FF    IC  DMem|ram~37582|dataa
    43.541      0.424 FF  CELL  DMem|ram~37582|combout
    43.950      0.409 FF    IC  G_MUX_JAL|\G_NBit_MUX:20:MUXI|o_O~0|datad
    44.075      0.125 FF  CELL  G_MUX_JAL|\G_NBit_MUX:20:MUXI|o_O~0|combout
    44.302      0.227 FF    IC  G_MUX_JAL|\G_NBit_MUX:20:MUXI|o_O~1|datad
    44.427      0.125 FF  CELL  G_MUX_JAL|\G_NBit_MUX:20:MUXI|o_O~1|combout
    47.406      2.979 FF    IC  G_REG|\regi:21:REG|\G_NBit_Reg:20:REG|s_Q|asdata
    47.807      0.401 FF  CELL  MIPSregister:G_REG|Nbit_reg:\regi:21:REG|dffg:\G_NBit_Reg:20:REG|s_Q

Data Required Path:
Total (ns)  Incr (ns)     Type  Element
==========  ========= ==  ====  ===================================
    20.000     20.000           latch edge time
    23.430      3.430  R        clock network delay
    23.438      0.008           clock pessimism removed
    23.418     -0.020           clock uncertainty
    23.436      0.018     uTsu  MIPSregister:G_REG|Nbit_reg:\regi:21:REG|dffg:\G_NBit_Reg:20:REG|s_Q
Data Arrival Time  :    47.807
Data Required Time :    23.436
Slack              :   -24.371 (VIOLATED)
============================================================================
```