Xiaotian Bai
N16340028

Computer Architecture: Final Exam

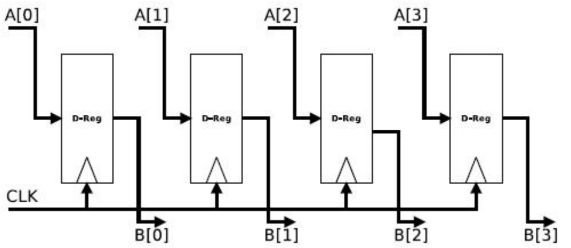
Part1

Question: Consider the 4 x 4 multiplier described in Digital Design and Computer Architecture, chapter 5 section 5.2.6 and answer the following questions:

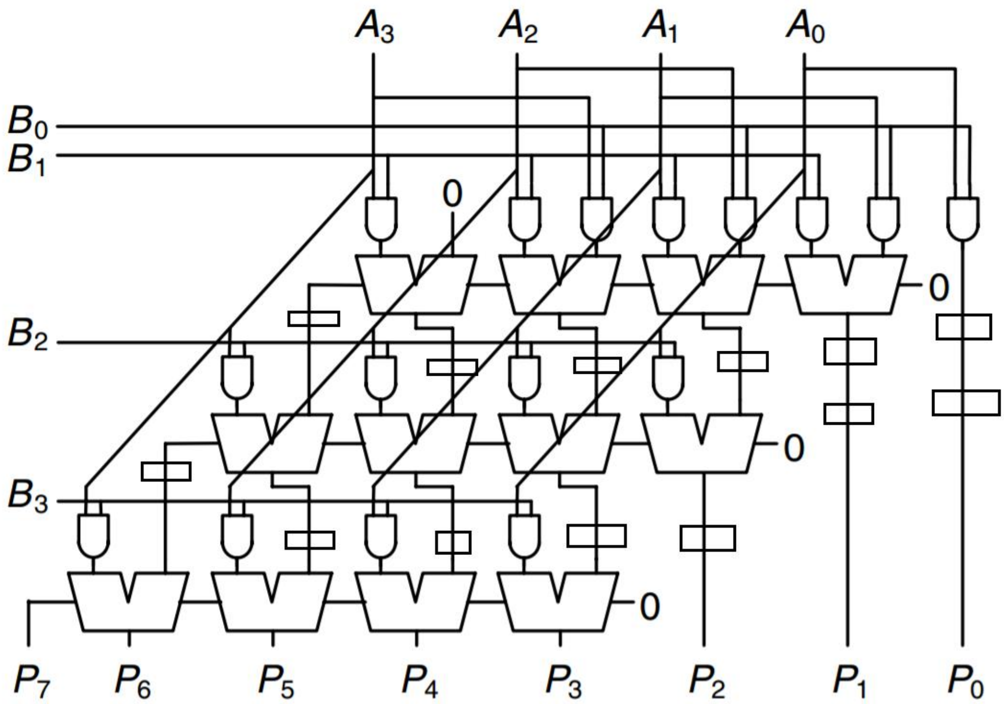
1. *Redesign the multiplier such that it is synchronize to a 1Ghz clock source.*
2. *Prove, by calculating the worse case combinatorial propagation delays, that your synchronize multiplier works reliably over 0 C to 100 C temperature range.*

Assume the following propagation delays:

Answer:
The delays peak at 100℃, so the redesign must works well at 100℃, where the gate delay is 310±10 ps and ALU delay is 600±10ps. Because the worst case delay must be smaller than 1000ps for 1GHz clock rate, the signal can at most transit through one gate plus one ALU in a clock cycle.
So we can put a D-Reg between each level consisting of one gate and one ALU



The new design with D-Regs is:



In the new design above, each rectangle represents an D-Reg, the multiplication needs 3 clock cycles, the result of P0~P7 will appear during the third clock cycle.
The worst case of propagation is one gate plus one ALU.

$$\text{Synchronize clock rate} \geq \frac{1}{310ps+10ps+600ps+10ps} = 1.075GHz$$. So the new design is qualified.

Part2

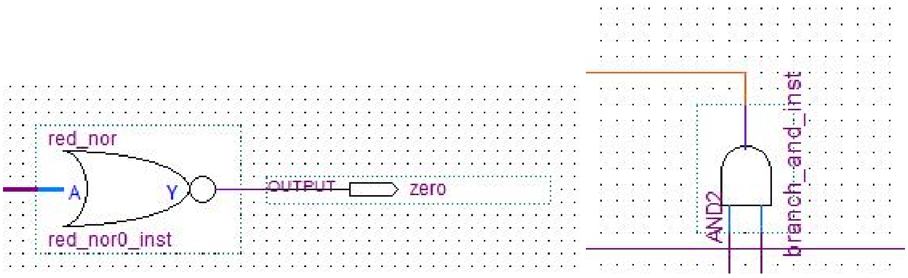
Question:Considering the MIPS SS v2 (Simple CPU) and its ISA for this part.
1.How does the simple CPU handle illegal instructions? Describe, with details, what Simple CPU does when an undefined opcode and/or funct code is presented.
2.Redesign Simple CPU's ALU to support the following instructions:
ble rx, ry, o↵set: pc += (rx < ry) ? (o↵set + 4) : 4
bge rx, ry, o↵set: pc += (rx > ry) ? (o↵set + 4) : 4
Note: You are only required to show modification to Simple CPU's ALU for question 2.

1.When an undefined opcode is presented, all the output of the "main_ctrl" module is 0. Because aluop[1..0]=00, we have alu_op[2..0]=010, thus the ALU has add function. Although ALU will add the two inputs, nothing in Register File or DRAM will change since the "regwrite" and "memwrite" signal is 0. The Program Counter will point to next instruction in IRAM, and CPU will continue processing next instruction.
When an undefined funct code is presented, the situation may be different. If the opcode is illegal, it is the same situation as discussed above. If the opcode is legal, for lw, sw and beq instructions, the funct code is irrelevant(don't care), so the function of CPU won't be affected.

Opcode	aluop	funct	ALU Func.	alu_op
lw:b"100011"	b"00"	XXXXXX	add	b"010"
sw:b"101011"	b"00"	XXXXXX	add	b"010"
beq:b"000100"	b"01"	XXXXXX	sub	b"110"

For other instructions(add, sub, and, or, slt), an undefined funct code will result in random outcome, the function of ALU will be unpredictable.

2.



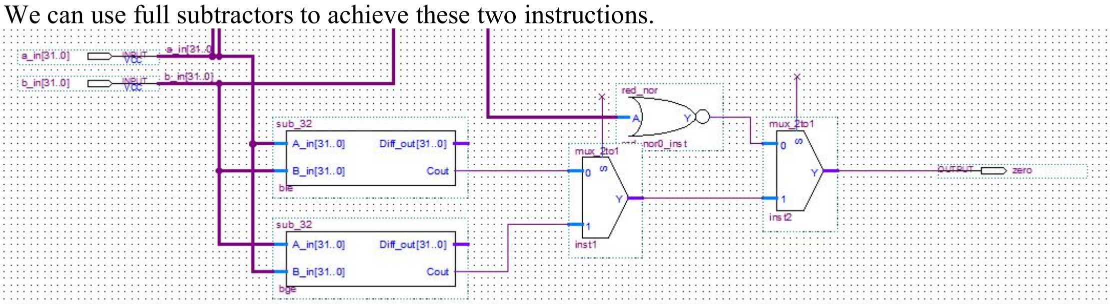
In simple CPU, "branch" and "zero" signals control the source of PC. If "branch"=1, when "zero"=1, pc+=(offset+4), when "zero"=0, pc+=4. We assume "branch"=1 for this question.
For ble rx, ry, offset, the "zero" signal of ALU should be

$$"\text{zero}" = \begin{cases} 1 & (\text{rx} < \text{ry}) \\ 0 & (\text{otherwise}) \end{cases}$$

For bge rx, ry, offset, the "zero" signal of AUL should be

$$"\text{zero}" = \begin{cases} 1 & (\text{rx} > \text{ry}) \\ 0 & (\text{otherwise}) \end{cases}$$

We can use full subtractors to achieve these two instructions.



As shown in the figure, the 32-bit full subtractors are used to compare rx and ry, because only rx<ry(ble) or rx>ry(bge) make "zero"=1, the Cout(borrow out) has to be 1 to make "zero"=1. Here I

used 2 muxes to separate ble, bge and beq. The mux control signal can be produced by logic gates with certain ALU_op code.
If the ALU processes ble, mux1=0, mux2=1. If bge, mux1=1, mux2=1. If beq, mux2=0

Part3

Question: Considering the MIPS SS v2 and its ISA for this part.
1.Assume the Simple CPU is pipelined, as described in lecture, what type of hazards you would encounter in Simple CPU. Remember you only have to consider instructions supported by Simple CPU.
2.Could a "NOP" inserter mitigate all hazards you listed above? Support your answer with details.

Answer:
1.There are 3 types of hazards that may happen in simple CPU, namely, Structural harzards, Data harzards and Control harzards.
(A)Structural harzards.
For example, the instructions can be described as

	CC1	CC2	CC3	CC4	CC5
add,sub,lw..	IM	Reg	ALU	DM	Reg
sw	IM	Reg	ALU	DM	
beq	IM	Reg	ALU		

If a register is used by two instructions at the same time, then Structural harzards arise. The situation can be:

CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
IM	Reg	ALU	DM	Reg*			
	IM	Reg	ALU				
		IM	Reg	ALU	DM		
			IM	Reg*	ALU	DM	Reg

As shown in the form above, Reg* can cause conflicts and hence Structural harzards.

(B)Data harzards.
For example, when the CPU processes the codes:
addi R4, R1, 1
addi R2, R4, 1

CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
IM	Reg1	ALU	DM	Reg4			
	IM	Reg4	ALU	DM	Reg2		

Because of pipelining, the value of R4 in "sub" instruction is not the outcome of the "addi" instruction, hence Data harzards occurs in this situation.

(C)Control harzards.
For example, when the CPU processes the codes:
beq R4, R3, 5
add R1, R2, R3

CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
IM	Reg3,4	ALU					
	IM	Reg2,3	ALU	DM	Reg1		

Because the "beq" instruction depends on the result of the comparison of R3 and R4, so only after CC3 can the CPU know where is the next instruction. If we don't insert NOP, the CPU will fetch the next instruction "add" before "beq" works, so Control harzards occurs.

2.
"NOP" inserters can mitigate all of the three harzards.
For each case we use the same example:
(A)Structural harzards.

CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
IM	Reg	ALU	DM	Reg*			
	IM	Reg	ALU				
		IM	Reg	ALU	DM	Reg	

			NOP	NOP	NOP	NOP	NOP
				IM	Reg	ALU	DM

Now there is no resource conflict, the Structural harzards has been mitigated.

(B)Data harzards.

CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
IM	Reg1	ALU	DM	Reg4			
		NOP	NOP	NOP	NOP	NOP	
			NOP	NOP	NOP	NOP	NOP
				IM	Reg4	ALU	DM

Now the second addi instruction can use the right value of Reg4, Data harzards has been mitigated.

(C)Control harzards.

CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
IM	Reg3,4	ALU					
	NOP	NOP	NOP	NOP	NOP		
		NOP	NOP	NOP	NOP	NOP	
			IM	Reg2,3	ALU	DM	Reg1

Now the CPU can fetch the next instruction for “beq” correctly. Control harzards has been mitigated.

Part4

Question: Consider the following code:
*1| char *a, *b, *c;*
2| <allocate 2^29 bytes of memory to a,b>
3| <allocate 2^12 bytes of memory to c>
4| for(i=0; i<=2^29; i++)
5| a[i] = b[i] + c[(i mod 2^12)];
Assume the following cache requirements :
DDR memory’s smallest transfer size is 512 bits or 64 bytes.
L2 cache is 16MB and is composed of 4096 4KB cache lines. A cache line is the smallest unit that could be transferred between DDR and L2 cache.
The memory subsystem is aligned to 4KB memory boundaries. Assume that memory allocation confirms to 4KB alignment.
Answer the following questions:
1.Find an optimal cache architecture (direct, n-way associative or fully associative) that would yield the best performance with lowest implementation complexity.
2.What type of replacement algorithm would yield the lowest miss rate?

Answer:
1.From the code we can see that a and b both have 2^29 bytes=512MB of memory, and c has 2^12 bytes=4KB of memory.
The point of the cache architecture is to avoid collision with lowest complexity. Direct mapped architecture is obviously unqualified, because it cannot avoid collision. Fully associative architecture will prevent the collision, but it’s too complicated and not economical.
The memory of c is exactly the memory of a cache line, so this cache line is always being used. Although c is small, because the program read data one byte a time, if c has collision with a or b, it will cause thousands of Cache Miss. So memory of a, b and c must be stored without competition.

I think the best architecture is 4-way set associative, where a, b, c can have their own cache lines within a set and don’t need to compete with each other. Also, the architecture is not complicated.

2.Because the cache line of C is always used, so we must Lest-Recently Used policy to protect memory of c from being replaced. With LRU policy, we will have the lowest miss rate.

Part5

Question:Consider Parallel BUS and Serial links described in I/O lecture for this part. State which I/O architecture (serial or parallel) is optimal for each of the following scenarios:
1.Lots of random 32-bit word transfers between CPU and I/O peripherals
2.Large amount contiguous burst transfers occur between I/O peripherals and a few burst transfer between I/O peripherals and CPU
3.Both burst transfers and few 32-bit word transfers between CPU and I/O peripherals.
You should support your answers with sufficient details.

Answer:
1.Parallel communication architecture is optimal for this case. Lots of word require a higher speed of data transferring. The random data won't cause interference in the wires. Moreover, the data is 32-bit, which is perfect in data alignment and synchronization for parallel transmission.

2.Between I/O peripherals, we need parallel communication to transfer large amount of data. Also, the burst transfer is contiguous, the noise or interference won't be significant.
Between I/O peripherals and CPU, we can use serial communication because the amount data is small and serial transmission won't cause interference.

3.Serial communication architecture is optimal for this case. Because the data contains burst transfer and 32-bit word, when the amount data change rapidly, the noise in parallel wires will be significant. In addition, the amount of data is small, so serial transmission is better.