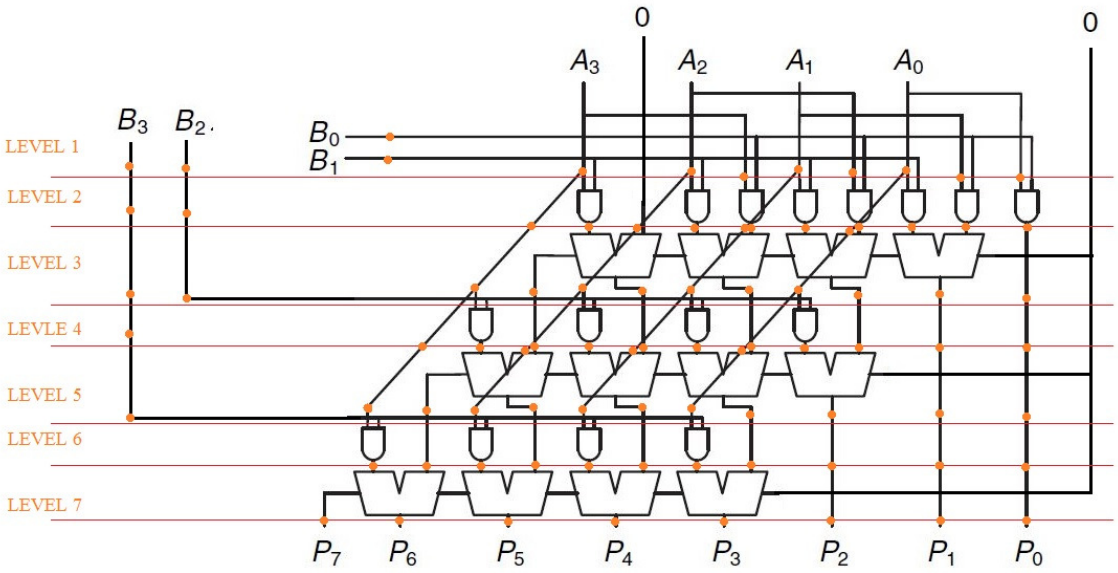
Sagar Panchal – N18051845
sap586

PART 1: DIGITAL DESIGN

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.**

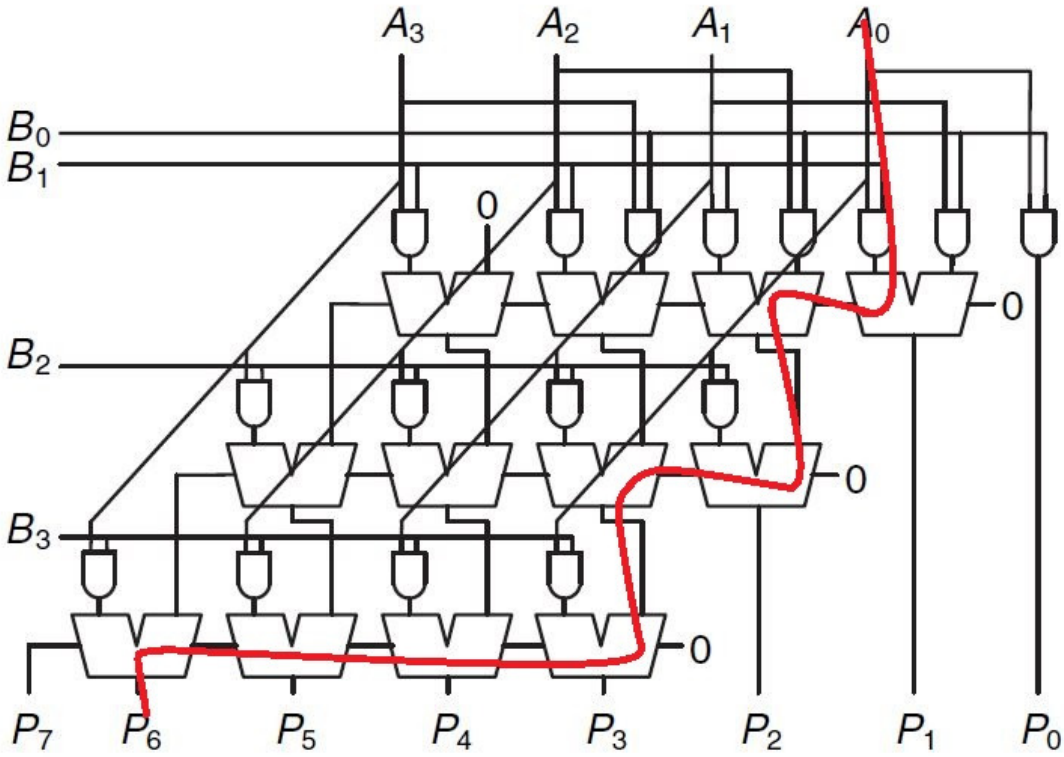
To synchronize the multiplier with 1GHz clock, we need to place the registers at each levels of combinatorial logic levels. These registers will be connected to the source of 1GHz clock. In order to do that, we will break the design into discrete logic levels. In the modified diagram below, we can identify the logic levels by placing the lines at the end of each levels. Then the registers are placed at each level which is denoted by an *orange dot*. These registers, when connected to the source of 1GHz clock will make the design to operate with the synchronization of 1GHz.



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:**

Temp. \ L.E.	Gate	ALU
100 °C	310ps ±10ps	600ps ±10ps
50 °C	260ps ±10ps	500ps ±10ps
0 °C	210ps ±10ps	400ps ±10ps

The worst case for the multiplier design would be the one with maximum numbers of ALUs and Gates in the path. As the delay of the ALU is more than that of the gates, the worst case would include the path with more number of ALUs. The diagram below shows the path with highest delay marked by red line.



The path marked with the red line includes 1 Gate and 8 ALU units. The calculation for all three temperature profiles given above is shown below:

0_C : (1*gate delay) + (8*ALU delay) = (1*220)+(8*410) = 3500ps
50_C : (1*gate delay) + (8*ALU delay) = (1*320)+(8*610) = 4350ps
100_C : (1*gate delay) + (8*ALU delay) = (1*320)+(8*610) = 5200ps

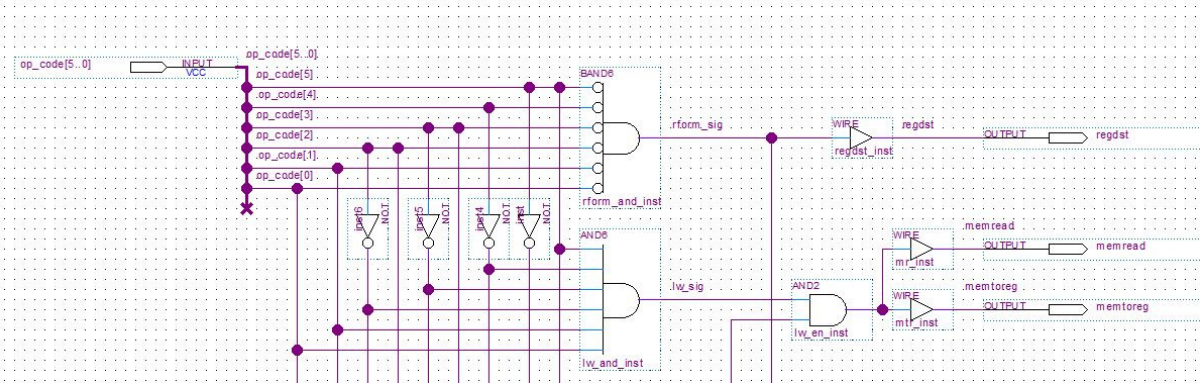
Sagar Panchal – N18051845
sap586

PART 2: SIMPLE CPU

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.**

The illegal instructions or undefined instructions are handled by the Control Unit in the MIPS SS v2 design. The Control Unit takes the opcode from the instruction and takes the control action respectively. The illegal/undefined instructions are controlled by logical AND gates which gives the desired output only when the opcode given is defined. The AND gates makes sure that all the 6 bits of the opcode are correct and takes the control decision only when all of them combines to form a right opcode.



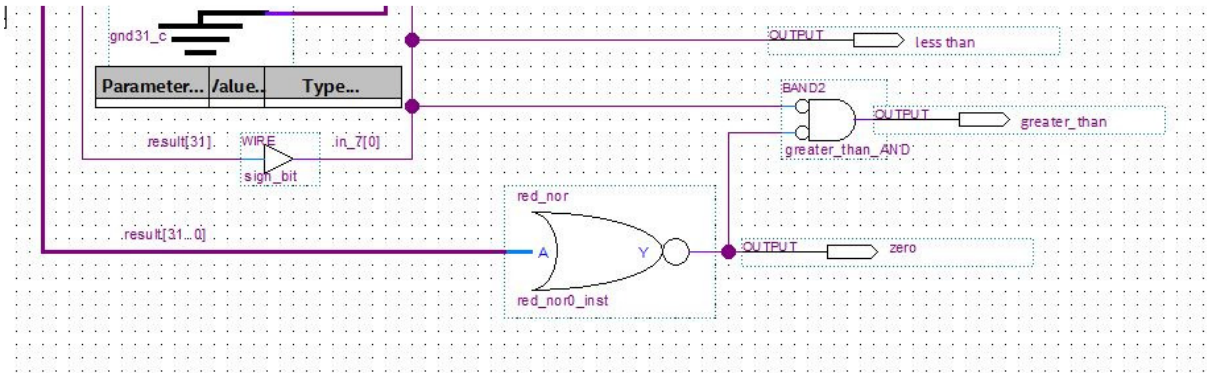
The above figure shows the 2 examples, one for R-type instructions and the other for lw (load word) instructions. The 6 bit AND gate will give the logical '1' if all the 6 bits of the opcode are correct. Even if 1 of the 6 bits is wrong, the AND gate will stay to logical '0'. The table below will gives the output list of the control unit when an illegal opcode is being detected by the Control Unit.

Control Unit output	Value
Regdst	0
Memread	0
Memtoreg	0
Memwrite	0
Branch	0
Alusrc	1
regwrite	1
Out_en	0
aluop	00

The alusrc value equal to '1' gives the value extended value of immediate address as one of the operand of the ALU unit. The value '1' of regwrite makes the 'wen' in the Register File to high which allows the register to be written.

2. **Redesign Simple CPU's ALU to support the following instructions:**
   - **ble rx, ry, offset: pc += (rx < ry) ? (offset + 4) : 4**
   - **bge rx, ry, offset: pc += (rx > ry) ? (offset + 4) : 4**

We will be using the 31st bit of the ALU result to decide whether the output is negative or positive. The 31st bit of the adder result will be '1' when the output is negative. We can use this to decide the whether the rx or ry is negative or positive. So I have used the 'less_than' flag directly as one of the output of the ALU. Moreover, if the 31st bit is not set, that means that the output is positive and we can use this logic to set the 'greater_than' flag. Also, I have used the value of zero flag in the logic of 'greater_than' flag. I have used the 2 bit AND gate with not gates to finally make the 'greater_than' flag high. If the 31st bit is not negative and the ALU output is not zero, that means the 'greater_than' flag is set.



In BLE instruction, if rx < ry, rx – ry will be negative and 31st bit of the adder will be high, so we will set the 'less_than' flag. In BGE instruction, if rx > ry, rx – ry will be positive and the 31st bit of the adder output will be low, and we will set the 'greater_than' flag. We will use these 'less_than' and 'greater_than' flags for the branch instructions.

Sagar Panchal – N18051845
sap586

PART 3: PIPELINED CPU

1. **Assume the Simple CPU is pipelined, as described in lecture, what type of hazards would you encounter in Simple CPU. Remember you only have to consider instructions supported by Simple CPU.**

Any conditions that prevents the execution of an instruction in designated clock cycle are known as hazards. There are three major types of hazards:

a. Structural hazards
b. Data hazards
c. Control hazards

Structural hazards are due to resource conflicts. When different instructions need to access the memory at a same time, there arise a condition of Structural hazard.
Data Hazards occurs due to dependencies in the program written. If there are instructions which need the data that is updated from the exact previous instruction, then there might be some problem with using the updated value.
Control Hazards usually occurs due to branch instructions. Consider the example of beq instruction. The next instruction to be executed is given by the PC and thus we need to wait till the value of PC gets updated.

2. **Could a "NOP" inserter mitigate all hazards you listed above? Support your answer with details.**

A "NOP" instruction can mitigate all the hazards related to the pipelining. But there can be always be a better option.
The Structural hazards can be solved by adding or modifying components in CPU. "NOPs" can be used where there are limitations in power or size requirements.
Data Hazards can be solved by forwarding module. Again, if hardware limitations are there, then Instruction re-ordering can mitigate the data hazards.
NOPs can help to mitigate the Control Hazards, but a better performance can be achieved by using Branch Prediction methods.

Sagar Panchal – N18051845
sap586

PART 4: MEMORY

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.**

Considering 4 KB Cache lines, the memory allocated to a, b, and c are as follows:
a : 2^29 = (2^17)*(2^12)
b : 2^29 = (2^17)*(2^12)
c : 2^12 = (1)*(2^12)
So, the best approach would be to use 3-way associative cache architecture as we would need to use the values of a, b and c. The 3-way associative architecture is used to store the variables a, b and c at different cache. This would be better rather than using the direct cache architecture where we use a complete single unit of 4096 4KB.

2. **What type of replacement algorithm would yield the lowest miss rate?**

In the code shown above, we will be using the value of 'c' more often than those of 'a' or 'b'. So, it would be profitable to store the values of 'c' in the cache without replacing it to improve hit-rate. Hence we can use Least Recently Used (LRU) algorithm to lower the miss rate.

Sagar Panchal – N18051845
sap586

PART 5: CONNECTIVITY AND TECHNOLOGY

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**

Parallel bus is best when we need to transfer 32 bits at a time. The case where a lots of 32-bits words are needed to transfer, the best I/O architecture to use is parallel bus.

2. **Large amount contiguous burst transfers occur between I/O peripherals and a few burst transfer between I/O peripherals and CPU**

Since a large amount of contiguous burst occurs between I/O peripherals, it would be better to use serial bus. If we use parallel bus, we would have to wait till 1 full word arrives to be transferred. In case of serial bus, we can just transfer a single bit as it arrives. In case of transfers between I/O peripherals and CPU, where few burst transfers occurs, it would be better to use parallel bus as we can send the whole word at a time.

3. **Both burst transfers and few 32-bit word transfers between CPU and I/O peripherals.**

Serial bus would be better in case of more number of burst transfers.