Problem 1: Cache Design

Consider a cache with the following parameters: \( N \) (associativity) = 2, \( b \) (block size) = 2 words, \( W \) (word size) = 32 bits, \( C \) (cache size) = 32 K words, \( A \) (address size) = 32 bits. You only need to consider word addresses.

(a) Show the tag, set, block offset, and byte offset bits of the address. State how many bits are needed for each field.

\[
C = S \times N \times b \implies S = \frac{C}{N \times b} = \frac{32 \text{ K words}}{2 \text{ blocks/set} \cdot 2 \text{ words/block}} = 8 \text{ K sets}
\]

Therefore there are \( \log_2(8 \cdot 1024) = 13 \) set bits.

The word size is 4 bytes. The number of bits required to address the blocks is

\[
\log_2(2 \text{ words/block} \cdot 4 \text{ bytes/block}) = 3 \text{ bits}
\]

with 1 bit for selecting the block (2 blocks per set) and 2 bits for selecting the byte (4 bytes per block).

The number of tag bits is then \( 32 - 3 - 13 = 16 \).

(b) What is the total size of all the cache tags in bits?

Each tag selects a block within a set, so we need \( 16 \cdot 32 \text{ K words} \cdot 1 \text{ block/2 words} = 256 \) Kbits of tags.

(c) Suppose each cache block also has a valid bit \( (V) \) and a dirty bit \( (D) \). What is the size of each cache set, including data, tag and status bits?
Solution:

Each cache block needs $1 + 1 = 2$ status bits, 16 tag bits, and $2 \cdot (4 \times 8) = 64$ data bits for a total of 82 bits for a block. There are 2 blocks for set, so each set needs a total of 164 bits.

(d) Design the cache using the building blocks in the figure below and a small number of two-input logic gates. The cache design must include tag storage, address comparison, data output selection, and any other parts you feel are relevant. Note that the multiplexer and comparator blocks may be any size ($n$ or $p$ bits wide, respectively), but the SRAM blocks must be $16 \text{ K} \times 4$ bits. Be sure to include a neatly labeled block diagram.

![Block Diagram](image)

Solution:

The design must use enough RAM chips to handle both the total capacity and the number of bits that must be read on each cycle. For the data, the SRAM must provide a capacity of 128 KB and must read 64 bits per cycle (one 32-bit word from each way). Thus the design needs at least $128 \text{ KB} / (8\text{ KB/RAM}) = 16$ RAMs to hold the data nad 64 bits / (4 pins/RAM) = 16 RAMs to supply the number of bits. These are equal, so the design needs exactly 16 RAMs for the data.

For the tags, the total capacity is 32 KB, from which 32 bits (two 16-bit tags) must be read each cycle. Therefore, only 4 RAMs are necessary to meet the capacity, but 8 RAMs are needed to supply 32 bits per cycle. Therefore, the design will need 8 RAMs, each of which is being used at half capacity.

With 8K sets, the status bits require another $8\text{ K} \times 4$-bit RAM. We use a $16\text{ K} \times 4$-bit RAM, using only half the entries.
Problem 2: FIFO Circuit

Your task is to design a 2-entry N-bit wide FIFO (queue) circuit. The circuit has interface signals as shown below. Unlike some FIFOs, on each clock cycle this version can perform only one operation—either read, write, or no operation. The operation is controlled by the \( \text{wr} \) and \( \text{rd} \) signals. The read operation is given priority over write. The three outputs \( \text{empty} \), \( \text{half} \), and \( \text{full} \), indicate whether the FIFO is empty, half-full, for full, respectively. A read operation when the FIFO is empty results in all zeros on the output. A write operation when the FIFO is full results in no operation. The \( \text{rst} \) signal is a synchronous control that results in the FIFO being empty.

Your design should take the form of a data-path and a FSM-based controller.

Using flip-flops, simple logic-gates and multiplexors, draw the data-path part of your design. You may assume that the individual flip-flops have clock enable (CE) inputs. Clearly label all input, output, and control signals (with names of your choice). Use hierarchy.
Draw a state transition diagram for the controller part of your FIFO design. Using the state transition diagram and additional Boolean expressions, if needed, indicate the logic for your control signals for the data-path (you don’t need to draw circuit diagrams for this part).

Solution:

```plaintext
empty = (STATE == 2'b00);
half = (STATE == 2'b01);
full = (STATE == 2'b11);

sel_empty = (STATE == 2'b00);
sel_buf = STATE[1];
WE = STATE + 1 // 01 if empty, 10 if half, 00 if full
```

---

**Problem 3: Unsigned Equality Checker Circuit**

Consider the design of a circuit that compares two unsigned integers. Specifically, the circuit takes two $N$-bit unsigned integers inputs, $A$ and $B$, and generates a single output bit $f$ equal to 1 iff $A > B$.

As you know, a subtrator circuit can be used for this function. However, a circuit optimized specifically for comparison is simpler. In this problem you will investigate different performance-cost tradeoffs for this optimized comparison function.

(a) Derive the simplest circuit to achieve this function. This circuit (similar to a ripple adder) will have $O(N)$ cost and delay $O(N)$.

Draw an instance of this circuit for $N = 4$. Show your work.

Solution:
(b) Derive a strategy for improving the performance of your circuit from part 1) to have $O(\log(N))$ delay and $O(N)$ cost. Draw a circuit to illustrate the structure of your circuit, for $N = 8$.

Solution:

Problem 4: Micro-architecture

Consider the design of a hardware accelerator for the following function described as a C program fragment:

```c
int S = 0;
```
for ( i=0; i<N; i++ ) {
    S = S + X[i]*Y[i];
    if ( S > K )
        { S = S-K; }
}

Both $X$ and $Y$ are arrays of 32-bit wide integers stored in a single 32-bit wide memory with a single 32-bit wide read port. $K$, $N$ is an integer values supplied to the accelerator at runtime. $S$ is a 32-bit wide value.

(a) In the space below sketch the datapath for a simple accelerator that minimizes the number of clock cycles. You can assume that the controller keeps track of the $i$ values and supplies $i$ to the datapath.

You can use common building blocks such as adders, subtractors, comparators, registers, etc. Clearly indicate the control signals and briefly describe the controller operation.

**Solution:**

![Datapath Diagram]

The `sel_XY` signal chooses between $X[i]$ and $Y[i]$ each cycle. The `sreg_en` is high once every other clock cycle.

(b) How many clock cycles will your accelerator take for $N$ loop iterations?

**Solution:**

Assume the memory asynchronous-read (zero cycle read latency). Solutions that assume synchronous-read will add 1 more cycle to account for the read latency.

Each loop iteration requires 2 accesses to the same memory ($X[i]$ and $Y[i]$), so it will require 2 cycles per iteration. Therefore it will take $2N$ cycles for $N$ loop iterations.

(c) Suppose now you can add more hardware to your design to speed up its operation by shortening the critical path. Is this possible? How?
Solution:
One solution is pipelining. We could divide the datapath into Fetch and Execute stages, as shown below.

(d) Is there a way to improve performance further? Explain.

Solution:
Adding a second read port on the memory would allow us to access \(X[i]\) and \(Y[i]\) in the same cycle per iteration, reducing the number of cycles per loop iteration from 2 to 1.

Problem 5: Computing n-degree Polynomial

Given 32-bit \(n+1\) input coefficients \(a_0, a_1, ..., a_n\) and a 32-bit input \(x\). We’d like to ask you to build a circuit to compute the \(n\) degree polynomial as follows.

\[
\text{out} = a_0 + a_1 \cdot x + a_2 \cdot x^2 + ... + a_n \cdot x^n
\]

Assume an 32-bit adder has a delay of 200 ps, and an 32-bit multiplier has a delay of 300 ps. You may use as many additional 32-bit registers, muxes, or logic gates as you like.

(a) Design your circuit such that it only uses \(n\) 32-bit adders and \(n\) 32-bit multipliers and takes one cycle to compute \(\text{out}\). Draw the circuit diagram. State clearly input and output signals. Hint: look for Horner’s method.

Solution:
Horner’s method for computing a polynomial rewrites the polynomial as follows:

\[
\text{out} = a_0 + x \left( a_1 + x \left( a_2 + x \left( a_3 + \cdots + x \left( a_{n-1} + xa_n \right) \right) \right) \right)
\]
(b) What is the delay of the critical path of your circuit (only considering the adders and multipliers)?

**Solution:**

There are \( n \) adders and \( n \) multipliers in the critical path from input \( a_n \) to \( \text{out} \).

\[
t_p = n \cdot t_{\text{add}} + n \cdot t_{\text{mult}} = n \cdot 500 \text{ ps}
\]

(c) Assume that your circuit takes a single 32-bit input coefficient each clock cycle. You may decide the order of input coefficients coming to your circuit. Design your circuit such that it only uses one 32-bit adder and one 32-bit multiplier to compute \( \text{out} \). Draw the circuit diagram. State clearly input and output signals.

**Solution:**

We can write the polynomial as a sum of lower degree polynomials:

\[
\text{out}_i = a_{n-i} + x \cdot \text{out}_{i-1}
\]

Where \( \text{out}_i \) is an \( i \)th degree polynomial and \( \text{out}_{i-1} \) is an \( (i - 1) \)th degree polynomial. We can pipeline this computation and store the value of \( \text{out}_{i-1} \) each cycle, multiply it by \( x \) and add it to the next input coefficient.
(d) What is the number of cycles it takes to compute the result and the delay of the critical path of your circuit in part (c)?

Solution:

The critical path only contains 1 adder, 1 multiplier and 1 register.

\[ t_p = t_{\text{add}} + t_{\text{mult}} = 500 \text{ ps} \]

(if you included the delays of the register):

\[ t_p = t_{\text{clk}} - Q \cdot t_{\text{add}} + t_{\text{mult}} + t_{\text{setup}} \]