CS220: Computer Organization
End-semester Examination Solution Sketch

- Please be concise in your answers and write legibly.
- Do not write unnecessarily long answers. Longer answers usually lead to lower scores.
- Every answer must be accompanied by adequate explanation. Any answer (correct or incorrect) without explanation will not receive any credit.
- This is an open-book/open-note examination.
- You can see your marks on 29th April at 6 PM in RM101.

1. A multiplication hardware implementing Booth's algorithm has one adder/subtracter and one left shifter. The adder/subtracter can complete an addition or a subtraction in one cycle. The shifter can complete a shift operation by any amount in one cycle.

1.(a) The two's complement representation of a positive even multiplier has two consecutive ones and all zeros. The positions of the ones are already known. How many cycles will Booth's multiplication hardware require to complete a multiplication by such a multiplier? **(3 points)**

Solution: Since the multiplier is positive and even, the ones are not at the edges. So, Booth's algorithm requires two left shifts, one addition, and one subtraction. This requires four cycles to complete. If the operations are pipelined, this would require three cycles (shift, shift and subtract, add).

1.(b) Now consider a positive even multiplier whose two's complement representation also has two ones and all zeros, but the ones are not consecutive i.e., there is at least one zero between the ones. The positions of the ones are already known. How many cycles will Booth's multiplication hardware require to complete a multiplication by such a multiplier? **(3 points)** Cleanly write down the steps of an algorithm that can complete the multiplication faster than Booth's algorithm in the case of such multipliers. **(6 points)** How many cycles will your algorithm require to complete a multiplication by such a multiplier? **(3 points)** You cannot introduce any new hardware and must work with only one adder/subtracter and one left shifter.

Solution: Booth's algorithm requires four left shifts, two subtractions, and two additions. This can be done in eight cycles. If the operations are pipelined, this can be done in five cycles (shift, subtract and shift, add and shift, subtract and shift, add). A better algorithm exploits the fact that if the bit positions i and j have ones, then the multiplication is equivalent to $(A << i) + (A << j)$ where A is the multiplicand. This can be done in three cycles.

2. The instruction mix of a program compiled for a multi-cycle non-pipelined processor is as follows: 15% store, 25% load, 15% control transfer, 35% integer arithmetic and logic, 5% shift, and 5% multiplication. The design is such that a load or a store takes two cycles, a control transfer instruction takes four cycles, an integer ALU instruction (excludes shift instructions) takes one cycle, a shift instruction takes four cycles, and an integer multiplication takes 25 cycles.

2.(a) Calculate the average CPI of this program when executed on this processor. **(2 points)**

Solution: Average CPI = $0.15 \times 2 + 0.25 \times 2 + 0.15 \times 4 + 0.35 \times 1 + 0.05 \times 4 + 0.05 \times 25 = 3.2$.

2.(b) The compiler can be optimized to replace each of 50% of the multiplication instructions by a sequence S of shift and add/subtract instructions such that the average length of S is five instructions. Out of these five instructions, x% are shifts and the remaining $(100 - x)$% are add/subtract instructions. Compute the maximum value of x such that this compiler optimization can reduce the overall execution time of the program by at least 10%. **(6 points)**

Solution: Let the number of instructions in the original program be n. So, total cycles = $3.2n$. The new cycles $\leq 3.2n \times 0.9$. The new cycles = $0.15n \times 2 + 0.25n \times 2 + 0.15n \times 4 + 0.35n \times 1 + 0.05n \times 4 + 0.025n \times 25 + 0.025n \times 5 \times (4x/100 + (100 - x)/100) = 2.575n + 0.125n(1 + 3x/100) \leq 3.2n \times 0.9$. Therefore, $x \leq 48$.

(2)

3. The instruction mix of a program compiled for a multi-cycle non-pipelined processor is as follows: 15% store, 25% load, 15% control transfer, 35% integer arithmetic and logic, 5% shift, and 5% multiplication. The design is such that a store takes four cycles, a load takes five cycles, a control transfer or ALU instruction takes three cycles, a shift instruction takes five cycles, and a multiplication instruction takes ten cycles.

(a) Calculate the average CPI of this program when executed on this processor. **(2 points)**

Solution: Average CPI = $0.15 \times 4 + 0.25 \times 5 + 0.15 \times 3 + 0.35 \times 3 + 0.05 \times 5 + 0.05 \times 10 = 4.1$.

(b) Doubling the number of registers in this processor leads to a reduction in the number of load and store instructions. Briefly explain why. **(2 points)**

Solution: Increasing the number of registers usually leads to less spills and fills because the compiler can now allocate more variables in registers.

(c) Doubling the number of registers in this processor also leads to an increased clock cycle time. Cite two possible reasons for this. **(2+2 points)**

Solution: First, increasing the number of registers can slow down register accesses lengthening the cycle time if a register read or write must be accommodated within a clock cycle. Second, increasing the number of registers requires more bits in the instruction to encode register numbers. This makes the instruction size bigger and can slow down the instruction fetch stage if one full instruction must be fetched in a cycle. This can also slow down the clock.

(d) Doubling the number of registers reduces the number of stores and loads by x each out of a total of 100 instructions i.e., the optimized program would have $(15 - x)$ stores and $(25 - x)$ loads if the original program had 100 instructions. However, incorporating a larger register file lengthens the cycle time of the processor by 10%. Compute the minimum value of x such that the optimized program running on the processor with double register count enjoys at least 10% reduction in execution time. All other instruction counts remain unchanged. **(6 points)**

Solution: Let the original instruction count be 100 and the cycle time be τ. Execution time = 410τ. New execution time $\leq 369\tau$. New execution time = $((15 - x) \times 4 + (25 - x) \times 5 + 15 \times 3 + 35 \times 3 + 5 \times 5 + 5 \times 10) \times 1.1\tau$. Therefore, $x \geq 820/99$.

-8

4. Consider including a new instruction known as branch on bit set in the MIPS ISA. The syntax of this instruction is: `bbs $1, pos, label`. It jumps to `label` if bit position `pos` in the contents of `$1` is one. Assume that `pos` is a compile-time constant meaning that the compiler knows its value. Further, assume that `pos` can take values starting from zero (which represents the least significant bit position).

4.(a) Recall that branch instructions are encoded in I-format (immediate format) of MIPS ISA. Suggest an encoding for `bbs` that fits the I-format. Assume a 32-bit data path. **(3 points)**

Solution: The register operand can be encoded using `rs` and `pos` can be encoded using `rt`. Note that in a 32-bit MIPS ISA, `pos` can take values in [0, 31]. The `label` can be encoded using the immediate field.

4.(b) If the design team decides to extend the data path to 64 bits, what problem does the above encoding face? How would you resolve it while keeping the instruction length unchanged? Is there any downside of your solution? **(1+2+2 points)**

Solution: With a 64-bit datapath, `pos` requires one extra bit. If the instruction length cannot be changed, there are two options. The first option is to borrow a bit from the immediate field. The downside of this solution is that the offset range of control transfer gets roughly halved. The second option is to introduce a new instruction `bbs64` which is used only when `pos` is more than 31. The new instruction implicitly encodes the one in the most significant bit of `pos`. This solution is possible only if there is at least one free opcode. There is no major downside of this solution.

4.(c) In the absence of the `bbs` instruction, how would you emulate `bbs $1, pos, label` using the existing 32-bit MIPS instructions? Suggest two MIPS assembly language code sequences one of which uses shift instruction while the other does not. **(4+6 points)** Use minimum possible number of instructions. Your code need not be syntactically correct. Hint: the best possible code sequence may differ depending on the value of `pos` being less than equal to 16 or bigger than 16.

Solution: Using shift:

```
srl  $2, $1, pos
andi $2, $2, 0x1
bne  $2, $0, label
```

Without shift (for pos less than 16):

```
andi $2, $1, mask // mask = 1 << pos prepared by compiler
bne  $2, $0, label
```

Without shift (for pos bigger than or equal to 16):

```
lui $2, mask // mask = 1 << (pos - 16) prepared by compiler
and $2, $1, $2
bne $2, $0, label
```

5. Consider the following segment of C code.

```
for(i=0;i<1000;i++) {
   if (i%2==0) {
      // Some statements
   }
   else if (i%3==0) {
      // Some statements
   }
}
```

5.(a) How many control transfer instructions would this code segment contain if it is translated to 32-bit MIPS ISA? **(3 points)** How many of them are forward branches and how many of them are backward branches? **(2 points)**

Solution: The pseudo-translation is shown below.

```
Loop:   $1 = i%2
        bne $1, $0, label1      // CTI1
        S1
        j label2                // CTI2
label1: $2 = i%3
        bne $2, $0, label2      // CTI3
        S2
label2: i++
        slti $3, i, 1000
        bne $3, $0, Loop        // CTI4
```

There are four control transfer instructions (three conditional and one unconditional). Three are forward branches and one is backward.

5.(b) When the translated code is executed, how many total control transfer instructions would be executed? **(4 points)** How many of them are forward branches and how many of them are backward branches? **(4 points)**

Solution: In each iteration, exactly three control transfer instructions would be executed. CTI1 and CTI4 are executed in each iteration. If `i` is even, CTI2 is executed. If `i` is odd, CTI3 is executed. So, a total of 3000 control transfer instructions are executed. Exactly 1000 of these are backward branches and 2000 are forward branches.

6. Consider a processor in which each instruction undergoes six stages of processing, namely instruction fetch (IF), instruction decode (ID), register file access (RF), execute (EX), data memory access (MEM), and register writeback (WB) with latencies $\tau_1, \tau_2, \ldots, \tau_6$ respectively such that $\tau_5 > \tau_1 > \tau_3 > \tau_4 > \tau_6 > \tau_2$.

6.(a) Consider a non-pipelined single-cycle instruction design of this processor where the IF stage begins fetching a new instruction on every posedge clock. How long does it take for this processor to complete a program of N instructions? **(2 points)**

Solution: Cycle time = $\sum_{i=1}^{6} \tau_i$ and each instruction takes one cycle to complete. So total execution time is $N \sum_{i=1}^{6} \tau_i$.

6.(b) Consider a non-pipelined multi-cycle instruction design of this processor where each stage takes one clock cycle. If all instructions must go through all stages before completing, how long does it take for this processor to complete the same program having N instructions? **(2 points)** Typically the clock frequency of such a design can be improved by subdividing certain stages into smaller stages increasing the overall number of stages. For this particular example, which stage should be subdivided for improving the clock frequency? **(2 points)**

Solution: Cycle time = τ_5 and each instruction takes six cycles to complete. So total execution time is $6N\tau_5$. The longest stage i.e., MEM should be decomposed to gain in clock frequency.

6.(c) Suppose in a new design, the IF and ID stages of the original six-stage design are fused to form a single stage of latency $\tau_1 + \tau_2$. The resulting five-stage processor is implemented using a non-pipelined multi-cycle instruction

design. All instructions must go through all stages before completing. Under what condition (on the τ_i values) will this new design be better than the old six-stage multi-cycle design? **(6 points)**

Solution: After fusing the IF and ID phases, each instruction takes five cycles to complete. The new cycle time is $\max(\tau_5, \tau_1 + \tau_2)$. So the required condition is $6\tau_5 > 5\max(\tau_5, \tau_1 + \tau_2)$. If $\tau_5 \geq \tau_1 + \tau_2$, this condition holds trivially. On the other hand, if $\tau_5 < \tau_1 + \tau_2$, the required condition becomes $\tau_5 > \frac{5}{6}(\tau_1 + \tau_2)$. Therefore, we conclude that the required conditions are $\tau_5 \geq \tau_1 + \tau_2$ or $\frac{5}{6}(\tau_1 + \tau_2) < \tau_5 < \tau_1 + \tau_2$. We can further combine these two conditions and conclude that as long as $\tau_5 > \frac{5}{6}(\tau_1 + \tau_2)$, the fusion is going to show improvement.

6.(d) If the original six-stage design is implemented in a pipelined fashion, how long will it take for this design to complete the same program having N instructions? Ignore all timing overheads in the flip-flops sitting between the pipeline stages. **(2 points)**

Solution: Cycle time = τ_5 and each instruction takes one cycle to complete. Ignoring the initial pipeline filling overhead, total execution time is $N\tau_5$.

7.(a) By inspecting the quotient of an unsigned division it is possible to infer the sequence of subtractions and additions that would have taken place if the division was done using the non-restoring division algorithm. Infer the sequence of subtractions and additions in the non-restoring unsigned division algorithm when the quotient is of length $2n$ and has the binary pattern $101010\ldots10$. In the inferred sequence, use 'S' and 'A' to represent a subtraction and an addition, respectively. **(3 points)**

Solution: The sequence is SSASASASA...SA where the length of the sequence is $2n + 1$.

7.(b) What pattern of the quotient would maximize the number of subtractions in the non-restoring unsigned division algorithm? Write all such quotients for a given length n of the quotient. **(3 points)**

Solution: The number of subtractions is maximized when the quotient is 111...1 or 111...10. In both cases, the number of subtractions is n.

7.(c) What pattern of the quotient would maximize the number of additions in the non-restoring unsigned division algorithm? Write all such quotients for a given length n of the quotient. **(3 points)**

Solution: The number of additions is maximized when the quotient is 000...0.

8.(a) A cache has 64-byte block size and 2048 blocks. Sketch an organization of this cache that would maximize the tag length. **(1 point)** What is the name of this organization? **(1 point)** Compute the tag length for this organization assuming address length to be 48 bits. **(3 points)**

Solution: Sketch is omitted. This is a fully-associative cache. Tag length = 48 − 6 = 42 bits.

8.(b) Sketch an organization of the aforementioned cache that would minimize the tag length. **(1 point)** What is the name of this organization? **(1 point)** Compute the tag length for this organization assuming address length to be 48 bits. **(3 points)**

Solution: Sketch is omitted. This is a direct-mapped cache. Tag length = 48 − 11 − 6 = 31 bits.

8.(c) Does the tag length have any influence on the search time for locating a block in the cache? If yes, explain how. If no, justify your answer. **(3 points)**

Solution: Tag length decides the width of the tag comparator directly influencing the time to locate a block in the cache.

8.(d) Consider a direct-mapped cache having 8-byte block size and 16 blocks. Assume that the cache is empty to begin with. The following sequence of addresses (presented in decimal) is sent to the cache: 20, 30, 22, 155, 40, 152, 145, 45, 20. Calculate the number of cache misses. **(6 points)**

Solution: Set indices for the addresses: 2, 3, 2, 3, 5, 3, 2, 5, 2. The accesses 22, 152, and 45 hit in the cache. The remaining six accesses miss.