(a) [Computer Arithmetic] (0.75pts) The chief architect of the MN-4363 processor, Tim FlimFlam, is exploring the ISA design space.

Tim claims that there is no need for a multiply instruction. A left shift should serve the purpose. One of the team members, Billy ShillyShally warns Tim that it is not left shift, but left shift along with add that would suffice to implement multiply.

- Would Tim's approach work? Explain.

- Would Billy's approach work? Explain.

- Demonstrate how two 16-bit numbers, $(129)_{10}$ and $(35)_{10}$ can be multiplied using Billy's technique.

(b) [ISA] (1.5pts) The chief architect of the MN-4363 processor, Tim FlimFlam, is exploring the ISA design space.

(i) Tim plans to restrict the instruction word to 32 bits, but wants to design a jump instruction that can change all bits of the PC. Their ISA already assumes byte-addressing and 32-bit wide general purpose registers (and PC). This is the most effective way to handle long jumps, Tim claims.

- Can Tim's approach work? Explain.

(ii) Tim's team member Billy ShillyShally, on the other hand, claims that a long jump can be implemented as a chain of MIPS-like pseudo-direct jumps and conditional branches.

- Can Billy's approach work? Are conditional branches required? Explain.

- How does Billy's "solution" compare to Tim's?

Under MIPS-type pseudo-direct addressing, the 26-bit immediate field of the (32-bit) jump instruction is interpreted as a word offset, hence multiplied by 4 (to arrive at an 28-bit constant), and concatenated with the higher order 4 bits of the current PC (to render an 32-bit address).

For conditional branches, the immediate field has only 16 bits, gets interpreted as a word offset (similar to jumps), and added to the current PC.

(c) [Branch Prediction] (1pt) The chief architect of the MN-4363 processor, Tim FlimFlam, is now working on the branch predictor. The processor relies on two separate structures: A prediction table to keep track of branch direction, and a BTB (Branch Target Buffer) to keep track of branch target addresses. Tim claims that, in processing a branch, if there is a miss in the BTB but a corresponding entry in the prediction table does exist, the instruction fetch should continue from PC+4, i.e. the fall-through path of the branch, since the predictor was likely wrong but BTB correct.

- Under what circumstances can this be the case? Provide at least two possible scenarios.
- Do you agree with Tim's solution? Explain.

Hint: BTB can be thought of as a cache, with limited number of entries. Branches constitute usually less than 20% of the instruction mix.

(d) [Quantitative Analysis] (0.75pts) As the chief architect of Massively Parallel Inc., you hired a summer intern called John Gustafson. He was in charge of characterizing the speedup of one of your Massively Parallel applications as a function of the number of processors, N, devoted to computation.
For N=1024, he claimed that he was able to observe a speedup of $\approx 512\times$ over the sequential version of the code (N=1). This result remains much larger than the speedup claimed by your previous intern, Gene Amdahl.
Assumption: The overall execution time of this code for N=1 would be s+p, where p corresponds to the section of code that can be parallelized. On N processors, this section would take $p/N <$ time units $>$. $s + p = 1 <$ time unit $>$. s does not change with N.
(i) Gene's result was in-line with the speedup Amdahl's Law suggests. Derive the speedup as a function of p for N=1024. What should p be to achieve a speedup $\approx 512\times$?
(ii) You find out that John approached the task in a different way: For Gene, the goal was reduction of (parallel) execution time for the *same* quantity of (total parallel) work (i.e. constant p). John, on the other hand, interpreted *speedup* as the increase in the (total parallel) quantity of work for the same (parallel) execution time. Further, he assumed that the quantity of work increases linearly with the number of processors N (and that the execution time is proportional to the quantity of work). If, for any N$>$ 1, $s_{Gustafson} + p_{Gustafson} = 1$, derive the speedup as a function of $p_{Gustafson}$ and N – *Gustafson's Law*[1]. What should $p_{Gustafson}$ be to achieve a speedup $\approx 512\times$?
(iii) You are now preparing a manual for your next intern. When should the new intern rely on Gustafson's Law, when on Amdahl's?

[1]"Gustafson Law" or "Gustafson-Barsis Law" indeed exists.