ECE 496 – Project Part 7
Assembly Challenge - 200 points

Must be submitted electronically by 1:15PM on Tuesday, April 23rd on Sakai

IMPORTANT: Please get started early! That way, when problems arise (which they will!), you will have time to ask me and/or the TAs for help.

1) Problem Statement

The subset sum problem is a classic problem in computer science that is emblematic of complexity theory and has some important applications in cryptography.

The problem asks: given a set of integers, is there a non-empty subset whose sum is zero? In this implementation, you will be given an array of up to 30 integers, with the input of the following form:

Number of elements

Elements separated by spaces

At the end you should print a subset of numbers that has zero sum, if such subset exists, or "NO" if no subset sum of zero exists. Note that if several subsets of zero sum exist, you just need to print one of them.

Examples:

```
Case 1:
Input:
6
1 2 3 4 -12 -10
Expected output:
1 2 3 4 -10
```

```
Case 2:
Input:
6
1 2 3 4 5 6
Expected output:
NO
```

2 Requirements

In this part of the project you will demonstrate that you can use your pipelined processor to run software that you write that solves the subset sum problem. As part of the project, you will download your (possibly enhanced) processor design into the FPGA prototyping board, demonstrate that it works during lecture, and show how fast it can solve the subset sum instances we give you.

Section 6 provides you with C code that solves the subset sum problem. Use the C code to understand how the problem works. Based on this understanding, you will write an assembly program (in the Duke 496/32 ISA) that solves the subset sum problem. Note that the program given is intentionally inefficient in the way it solves the problem and that there are multiple ways of optimizing it when you write your version.

3 The Need For Speed

The first two groups whose demos are the fastest (and correct!) will be awarded bonus points: 50 points for 1st place and 25 for 2nd. Also, up to 10 bonus points could be assigned to any group that implements a particularly interesting feature, even though their demo is not one of the fastest three demos, up to the discretion of the professor and the TAs.

You are free to make the processor go faster using either or both of the following two approaches. You do not have to do any of this, though. You will get full credit if your processor correctly solves the problem.

- Use the currently unused opcodes in the ISA: there are several opcodes intentionally unused. You can use these opcodes for any purposes you want. You will want to slightly modify the assembler to recognize these opcodes and generate appropriate code.

- Enhance the microarchitecture: If you want to add branch prediction, more pipe stages, a 2-wide pipeline, or any other microarchitectural feature, you may do so.

Whatever you do, though, must still comply with the ISA specification. Remember: speed is great, but correctness is the most important feature.

4 Hardware Demos

As with the previous part of the project, we will have graded demos of the FPGA implementations that will be held in lab (Hudson 202A) during class time on Tuesday, April 23rd. You will want to test your designs on the hardware long before then, to make sure they work correctly (even if they worked without problem in Quartus).

5 Submission

Submit two files for this assignment: `newprocessor.qar` (the processor design) and `software.s` (the software) in Sakai before 1:15pm on Tuesday, April 23rd.

6 C Code for Subset Sum

```
#include <stdio.h>
#define MAX 30

// function for calculating the subset sum of the array
vals.
// extremely naive implementation that tests all possible
subsets until one sums to zero.
// on a 32-bit processor it works for up to 30 elements in
vals.
int subsetSum(int n, int vals[]){
    int currPerm = 1;
    int lastPerm = 1 << n;
    int i;
    int sum;

    while(currPerm < lastPerm){
         sum = 0;
         for(i = 0; i < n; i++){
              if (currPerm & (1 << i))
```

```
                    sum += vals[i];
          }
          if(sum == 0){
               for(i = 0; i < n; i++){
                    if (currPerm & (1 << i))
                         printf("%d ", vals[i]);
               }
               printf("is a solution\n");
               return 1;
          }
          currPerm++;
     }
     printf("No solution.\n");
     return 0;
}

int main(int argc, char** argv){
     int n;
     int vals[MAX];
     int i;

     printf("N: "); //number of elements
     scanf("%d", &n);
     printf("Elems: "); //elements separated by spaces
     for(i = 0; i < n; i++){
          scanf("%d", vals+i);
     }

     subsetSum(n,vals);
     return 0;
}
```