

Computer Organization Homework #6 Grading Standard

You should do the following optimizations as the table shows.

1. About the first function, since Prof. Hollinger had changed the function substr(), if you gave two or more reasonable optimizations of the function substr(), not necessarily the same as the followings, that is OK.
2. Fail to give one optimization method was given at least 3 points off. And if you could not give even one reasonable method, you will be given 9 to 33 points off based on your explanations.

	Loop Unrolling or Parallelism	Pointer Code	Unneeded Mem Reference	Return Early (inefficiency)
1 substr()	X			X
2 transpose()	X	X		
3 multiply()	X	X	X	

3. If your code is too short to express the necessary optimization, 2 points off.
4. For each function, no source code result in 5 points off.
5. For each function, if the optimization changed the semantics, 5 points off.
6. Optimized but cannot provide explanation for any optimization, 11 points off.
7. Totally no optimization for a function, we gave 33 points off.

Here are some samples of the optimization code (part or one step of them). About the loop unrolling and pointer code, you may refer to the textbook. If you still have questions about the grading, feel free to contact us, or drop in at our office hours.

Return Early and Loop Inefficiency Optimizations Sample:

```
#include <string.h> /* needed for strlen */
/* Is p a substring of s?
   returns 1 if p is a substring of s,
   otherwise returns 0
*/
int substr(char s[], char p[]) {
    int i;
int flag=0; /* assume we don't find p in s */

    len1=strlen(s);
    len2=strlen(p);
    /* look for the substring at each possible starting point. */
    for (i=0;i<len1-len2+1;i++) {
        if (strncmp(&s[i],p,len2)==0) {
            // we found a match, set flag to 1
            return 1;
        }
    }
    return 0;
}
```

Unneeded Memory Reference Optimization Sample:

```
/* use whatever size you want, but don't use something too small
   or your measurements will probably not be reliable */

#define SIZE 100
typedef int mtx[SIZE][SIZE];

/* matrix multiplication. c=ab
   assumes that all of the matrices are the same
   size (and square).
*/
void matrix_mult(mtx a, mtx b, mtx c) {
    int i,j,k;

    /* initialize c to all zeros */
for (i=0;i<SIZE;i++)
for (j=0;j<SIZE;j++)
c[i][j]=0;

    /* do the multiplication */
    for (i=0;i<SIZE;i++) {
        for (j=0;j<SIZE;j++) {
            sum = 0;
            for (k=0;k<SIZE;k++) {
                sum += a[i][k]*b[k][j];
            }
            sum = c[i][j];
        }
    }
}
```