Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InES Institute of
Embedded Systems

TSM_EmbHardw 2015
Exercises Week 9 / Lecture 2

Exercise Set A

Project 3: Learn to use the GNU profiling tools and cachegrind and valgrind and use them to analyse performance of the Sobel algorithm. Use loop optimisations to improve the runtime of the code. Systematically test the –O and the –O2/-O3 compiler settings. Which optimisations are applied to your code and what improvements do they bring?

Project 4: Get the custom instruction hardware to work. Optimise processing of the Sobel algorithm by using a custom instruction.

Exercise Set B

Exercise 1: Loop Unswitching:

The following code

```
int i, w, x[1000], y[1000];
  for (i = 0; i < 1000; i++) {
    x[i] = x[i] + y[i];
    if (w)
      y[i] = 0;
  }
```

Is unswitched to

```
int i, w, x[1000], y[1000];
  if (w) {
    for (i = 0; i < 1000; i++) {
      x[i] = x[i] + y[i];
      y[i] = 0;
    }
  } else {
    for (i = 0; i < 1000; i++) {
      x[i] = x[i] + y[i];
    }
  }
```

Compile both versions and – by analysing the assembler code – determine the speedup achieved by unswitching

Exercise 2: Loop Hoisting

Analyse the following code and optimise – don't use loop unrolling

```
void loop_invariants1(struct obj *obj,  Uint32 *xpos,
                      Uint32 *ypos, Uint8 two_dimensions, Uint32 t)
```

TSM_EmbHardw 2015
Exercises Week 9 / Lecture 2

```
{
  Uint32 i;


  for (i=0; i<MAX; i++)
  {
          obj[i].x = (5 * obj[i].x * t * t) + xpos[i]);
          if(two_dimensions)
          {
                  obj[i].y = ypos[i] * t;
          }
  }
}
```

Exercises: Exam Preparation

Exercise 1:

Here is some code that is supposed to multiply 2 matrices:

```
matrixmult(a,b,c)

float a[4][4], b[4][4], c[4][4];

{
    int x, y;
    float temp[4][4];


    for(y=0; y<4 ; y++)
        for(x=0 ; x<4 ; x++) {
            temp[y][x] = b[y][0] * a[0][x]
                       + b[y][1] * a[1][x]
                       + b[y][2] * a[2][x]
                       + b[y][3] * a[3][x];
```

TSM_EmbHardw 2015
Exercises Week 9 / Lecture 2

```
        }
    for(y=0; y<4; y++)
        for(x=0; x<4; x++)
            c[y][x] = temp[y][x];
}
```

Comment on this code and the application of loop unrolling, in-lining, custom instructions and cache-aware programming.

Exercise 2:
Given the program code below. Assume that all instructions execute in a single processor cycle (including the JLT instruction !).

C-code :		***Assembly code :***
short sum=0;		MOVE R5,#0
for (i=0;i<10000;i++) {		MOVE R0,#0
sum += a[i]*b[i];		MOVE R1,#&a[0]
		MOVE R2,#&b[0]
	loop:	LOAD R3,R0[R1]
		LOAD R4,R0[R2]
		MUL R3,R3,R4
		ADD R5,R5,R3
sum /= 2;		ASR R5,R5,#1
}		ADD R0,R0,#1
		CMP R0,#10000
		JLT loop
c=sum;		MOVE R6,#&c
		NOP
		STORE 0[R6],R5

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

InES Institute of
Embedded Systems

TSM_EmbHardw 2015
Exercises Week 9 / Lecture 2

a.) Mark with a circle all assembly instructions that are removed when we use loop-unrolling.
b.) Mark with a square all assembly instructions that can be combined in a custom instruction.
c.) How many processor cycles are required to execute this program code?
d.) When using loop-unrolling, how much do we speed up this program code?
e.) When using loop-unrolling, how many assembly instructions do we add to the code?
f.) When using custom instructions, how much do we speed up this program code?
g.) When using custom instructions, how many assembly instructions do we add to/remove from the code?
h.) Discuss which method would be best for this code segment :
i.) Examine the assembler code and show where allocation, binding and scheduling have been applied.