Code Optimization[1]

Instruction Pipelining

- Branching can be a bit of a problem for most CPUs while executing instructions in a pipelined fashion
- Simple branch example

  ```
  if ( test )
      value++;
  ```

  - When CPU encounters a branch, it may not have enough time to finish evaluating the test before it is time to decide whether to branch or not
  - Processor can only guess the instructions to fetch next
  - Incorrect guess leads to dismantling all operations currently in progress, and restart in correct place
- Predictable rules for guess
  - If branch jumps forward (if statement), it is assumed not to be taken
  - If branch jumps backward (loop), it is assumed to be taken
    * Loops tend to iterate more than once
  - Adding if ... else statement to code
    * *Place most common code after the if*
    * *Place rarely used code after the else*
- `if ... else` statements are concerned with only single data stream
  - Impossible to write code that can be pipelined or executed in parallel
  - Code with branching operates much slower than branchless code to do the same thing
  - Find a way to get rid of branches and write algorithms that work for all possible inputs without special cases
  - Example code to convert an array of `long`s to an array of `short`s with clipping
  - Simple version of function

    ```
    void convert ( SInt32 * src, SInt16 * dest, UInt32 sample_count )
    {
        SInt32 value;
        while ( sample_count-- )
        {
            value = *src;
            if ( value > SHRT_MAX )
                value = SHRT_MAX;
            else
                if ( value < SHRT_MIN )
                    value = SHRT_MIN;
            *dest = value;
            src++;
            dest++;
        }
    }
    ```

  - Branchless version of the same code

[1]Most of the material in this set of notes is from the AltiVec Tutorial by Ian Ollmann

```
void convert ( SInt32 * src, SInt16 * dest, UInt32 sample_count )
{
    SInt32 value;
    while ( sample_count-- )
    {
        value = *src;
        sign = value >> 31;
        value ^= sign;
        value = ( value | ( ( 0x7FFF - value ) >> 31 ) ) & 0x7FFF;
        value ^= sign;
        *dest = value;
        src++;
        dest++;
    }
```

- Version of the same code with limited branching

```
void convert ( SInt32 * src, SInt16 * dest, UInt32 sample_count )
{
    SInt32 value;
    while ( sample_count-- )
    {
        value = *src;
if ( value != SInt16 ( value ) )
        {
            value >>= 31;
            value ^= 0x7FFF;
        }
        *dest = value;
        src++;
        dest++;
    }
```

  * Branchless version is 4% faster in worst-case scenario where most of the data needed to be clipped
  * In best case scenario where lass than half needed to be clipped, limited branch code performed about 50% faster