Chapter 7
Linear Assembly

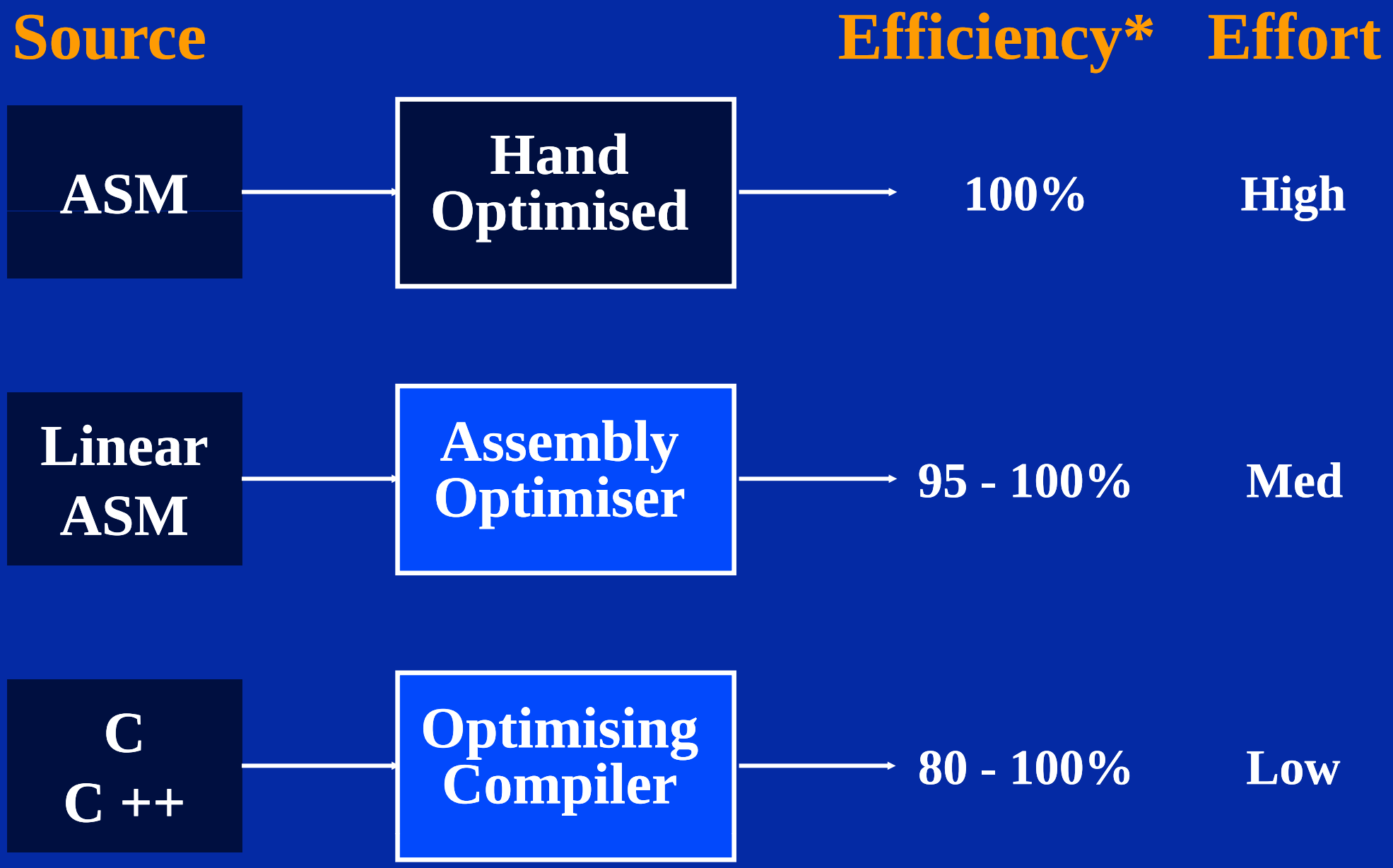
Learning Objectives

- Comparison of programming techniques.
- How to write Linear Assembly.
- Interfacing Linear Assembly with C.
- Assembly optimiser tool.

Introduction

- With the assembly optimiser, optimisation for loops can be made very simple.
- Linear assembly takes care of the pipeline structure and generates highly parallel assembly code automatically.
- The performance of the assembly optimiser can easily reach the performance of hand written assembly code.

Comparison of Programming Techniques



*** Typical efficiency vs. hand optimized assembly.**

Writing in Linear Assembly

- **Linear assembly is similar to hand assembly, except:**
  - **Does not require NOPs to fill empty delay slots.**
  - **The functions units do not need to be specified.**
  - **Grouping of instructions in parallel is performed automatically.**
  - **Accepts symbolic variable names.**

```
        ZERO    sum
loop    LDH     *p_to_a, a
        LDH     *p_to_b, b
        MPY     a, b, prod
        ADD     sum, prod, sum
        SUB     B0, 1, B0
        B       loop
```

How to Write Code in Linear Assembly

- **File extension:**
  - Use the ".sa" extension to specify the file is written in linear assembly.
- **How to write code:**

```
_sa_Function    .cproc

                ZERO    sum

loop            LDH     *pm++, m
                LDH     *pn++, n
                MPY     m, n, prod
                ADD     sum, prod, sum
     [count]    SUB     count, 1, count
                B               loop

                .return sum
                .endproc
```

.cproc defines the beginning of the code

NO NOPs required
NO parallel instructions required
NO functional units specified
NO registers required

.return specifies the return value

.endproc defines the end of the linear assembly code

Passing and Returning Arguments

- "pm" and "pn" are two pointers declared in the C code that calls the linear assembly function.
- The following function prototype in C calls the linear assembly function:

```
int y = dotp (short* a, short* x, int count)
```

- The linear assembly function receives the arguments using .cproc:

```
_dotp   .cproc    pm, pn, count
        ...
        .return   y
        .endproc
```

Declaring the Symbolic Variables

- All the symbolic registers except those used as arguments are declared as follows:

```
.reg   pm, pn, m, n, prod, sum
```

- The assembly optimiser will attempt to assign all these values to registers.

Complete Linear Assembly Code

```
_dotp           .cproc pm, pn, count
                .reg m, n, prod, sum

                ZERO    sum

loop            LDH     *pm++, m
                LDH     *pn++, n
                MPY     m, n, prod
                ADD     sum, prod, sum
     [count]    SUB     count, 1, count
                B               loop

                .return sum
                .endproc
```

- **Note: Linear assembly performs automatic return to the calling function.**

Function calls in Linear Assembly

- In linear assembly you can call other functions written in C, linear assembly or assembly.
- To do this the .call directive is used:

Function1.sa

```
_function1   .cproc a, b
             .reg y, float1

             MPY  a,b,y

             .call float1 = _fix_to_float(y)
             .return
             .endproc
```

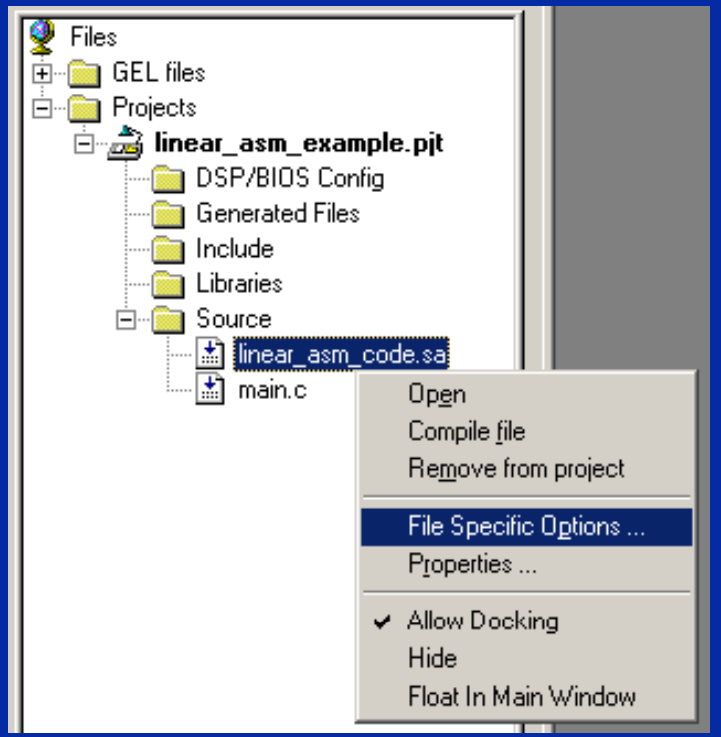
Fix_to_float.sa

```
_fix_to_float     .cproc fix
                  .reg float1

                  INTSP fix, float1

                  .return float1
                  .endproc
```

- Note: Branch out of a linear assembly routine is not allowed.

Invoking the Assembly Optimiser

- The development tools recognise the linear assembler code by the file extension ".sa".
- The assembly optimiser uses the same options as the optimising C compiler.
- Note: In CCS you can change the options of each file individually by right clicking on the file in the project view and selecting "File Specific Options…".



Linear Assembly Examples

- **The following chapters have code written in linear assembly:**
  - **\Code\Chapter 15 - Infinite Impulse Response Filters**
  - **\Code\Chapter 17 - Goertzel Algorithm**
- **For more details on Interfacing C and Assembly see Chapter 11.**

Chapter 7
Linear Assembly
- End -