Advanced Computer Architecture

—

Part I: General Purpose
Trimaran Practical Lab (II)

EPFL – I&C – LAP

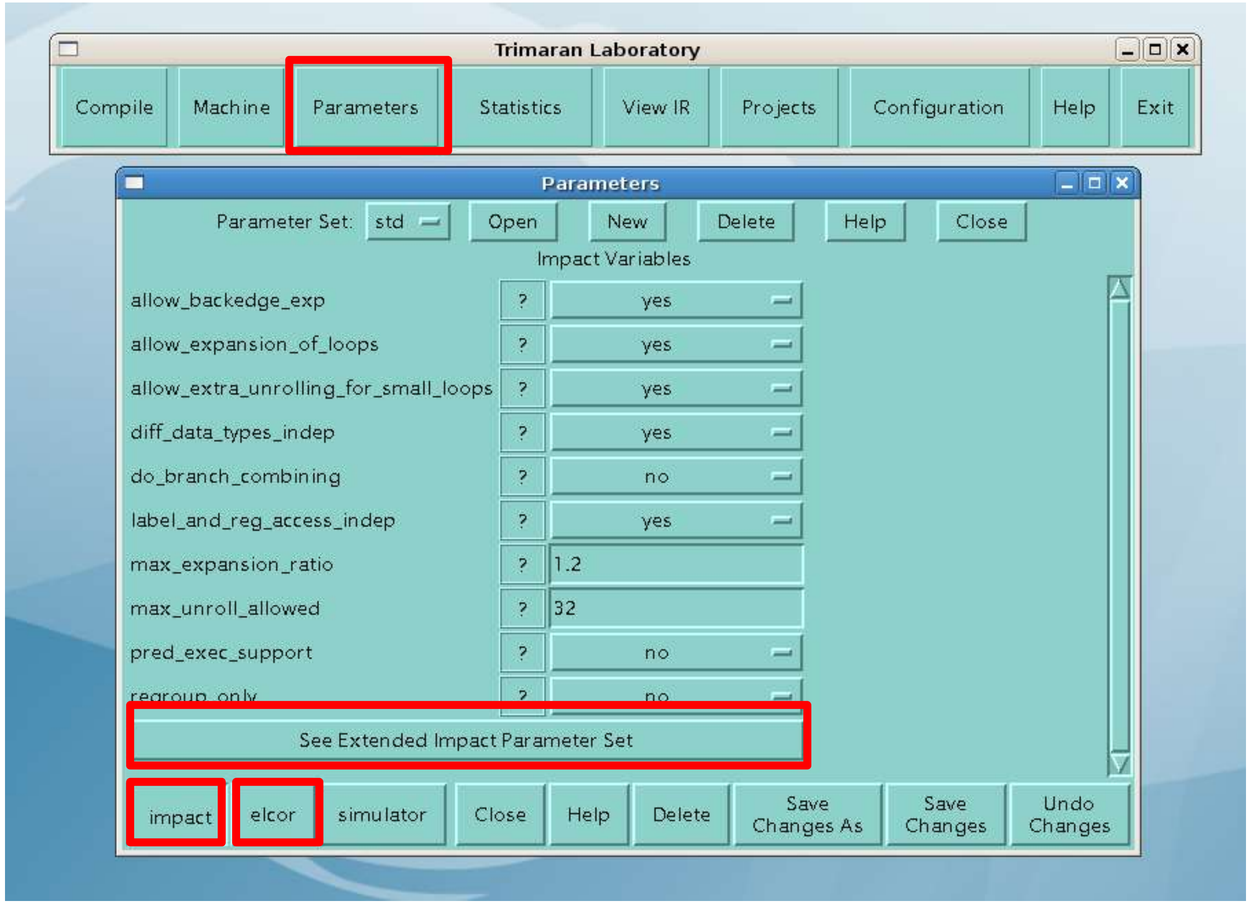
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Exploiting ILP Statically

- VLIW requires optimized software for performance
- Different techniques used by compilers
  - Predication
  - Loop Unrolling
  - Software pipelining/Modulo scheduling
- Lab 3 explores loop unrolling
- This lab explores predication and modulo scheduling

Compiler Optimizations

- We can control the Trimaran compiler optimizations
- Allows optimization isolation and effect comparison



allow_backedge_exp	?	yes
allow_expansion_of_loops	?	yes
allow_extra_unrolling_for_small_loops	?	yes
diff_data_types_indep	?	yes
do_branch_combining	?	no
label_and_reg_access_indep	?	yes
max_expansion_ratio	?	1.2
max_unroll_allowed	?	32
pred_exec_support	?	no
regroup_only	?	no

Predication

- We explore predication with if-then benchmark
- Benchmark source code

```
~/trimaran-workspace/benchmarks/ifthen/src/
```

- To observe predication effects, we will turn off both loop unrolling and modulo scheduling optimizations

Predication Setup

- Open “Parameters” “std”

- Under “impact”, open “Extended Impact Parameter Set”
- Turn off loop unrolling

```
do_loop_unroll = no
```

- Under “elcor”, open “DRIVER_DEFAULTS”
- Turn off modulo scheduling

```
do_modulo_scheduling = no
```

- “Save Changes As” “std_predication”

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Predication Compilation

- Compile ifthen benchmark with predication
  - Open “Compile”
  - Benchmark “ifthen”
  - Parameter-Set “std_predication”
  - Project-File “NEW” “ifthen”
  - Compilation and Simulation
    - Basic Blocks and Hyper Blocks
    - Code Generation
    - Processor Simulation
  - GO!

Predication

- Compare the two assemblies generated
- How does the CMPP instruction affect performance? Relate to the instruction latency.
  - What architectural support enables CMPP instruction?
- Compare the number of cycles, number of operations, size of code, and IPC

Modulo Scheduling

- We explore modulo scheduling with “modulo_scheduling” benchmark
- Benchmark source code

```
~/trimaran-workspace/benchmarks/modulo_scheduling/src/
```

- We will compare execution with and without modulo scheduling
  - Note: We will first have to add an extra functional unit just for the purpose of the experiment

Adding a Functional Unit

- Edit the machine used to have 3 floating point units
  - Open “Machine” and edit “hpl_pd_elcor_std.hmdes2”
  - In the text editor, edit line 166

```
$def !float_units 3
```

  - Close the editor and “Make”
- Functional unit latencies

Int_alu_latency	1
Float_alu_latency	3
Load_localmem_latency	2

Modulo Scheduling Compilation

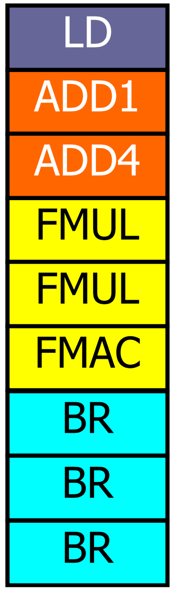
- Compile modulo_scheduling benchmark
  - Open “Compile”
  - Benchmark “modulo_scheduling”
  - Parameter-Set “std”
  - Project-File “NEW” “modulo_scheduling”
  - Compilation and Simulation
    - Basic Blocks
    - Code Generation
    - Processor Simulation
  - GO!

Unoptimized Assembly Loop

```
r2 = _NUM
…
r33 = 0
r3 = ld r2
```

```
r2 = _a
```

BB7:



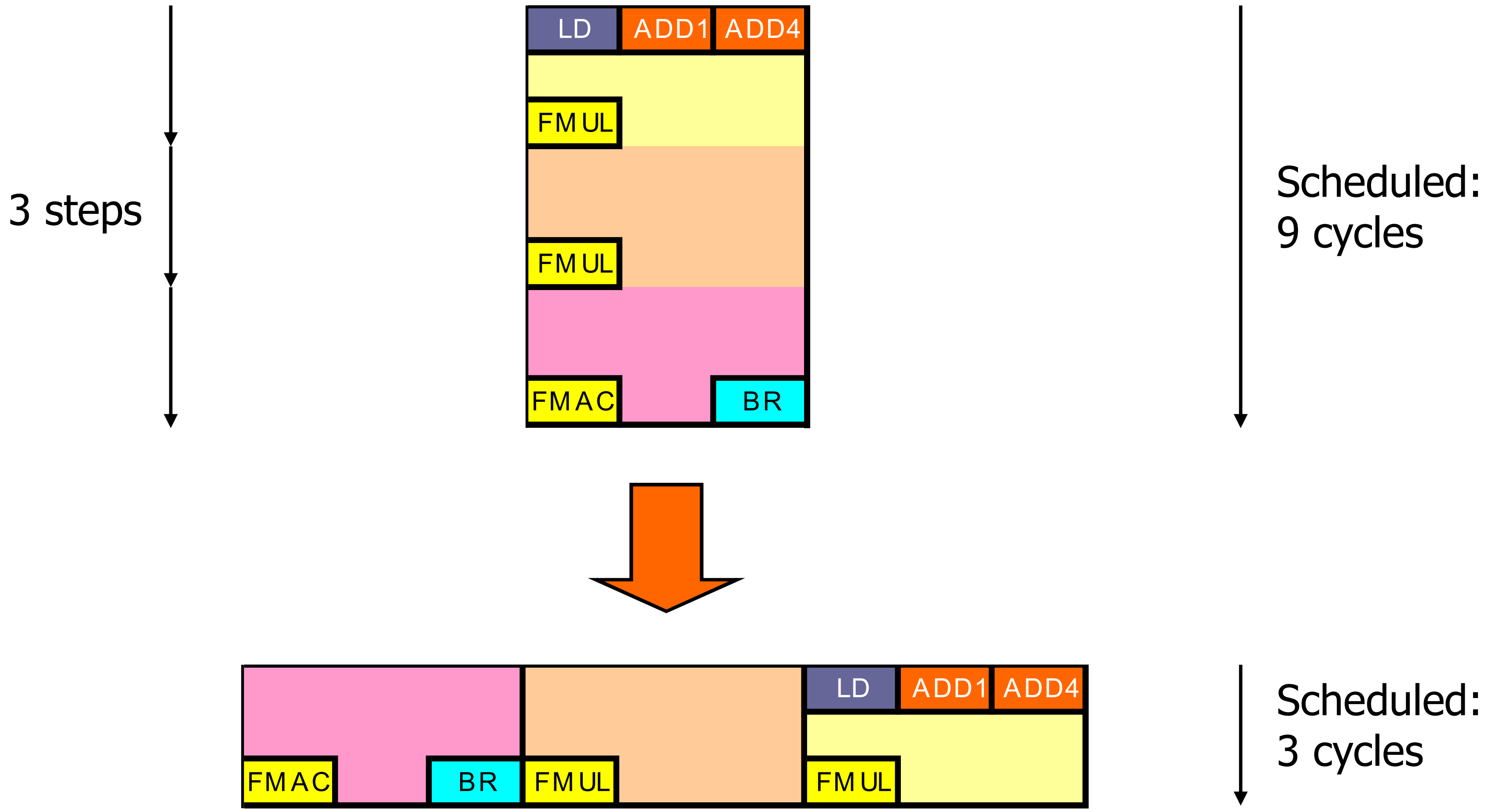
```
r2 = ld r2
r33 = r33 + 1
r2 = r2 + 4
r4 = r2 * r2
r5 = r4 * r2
r3 = *+ r5 r2 r3
r2 = pbrr BB7 0
p2 u = cmpp.<.UN.UN r33 r3
brct r2 p2
```

Ri : General Purpose Reg.

Ri : Floating Point Reg.

Ri : Branch Target Reg.

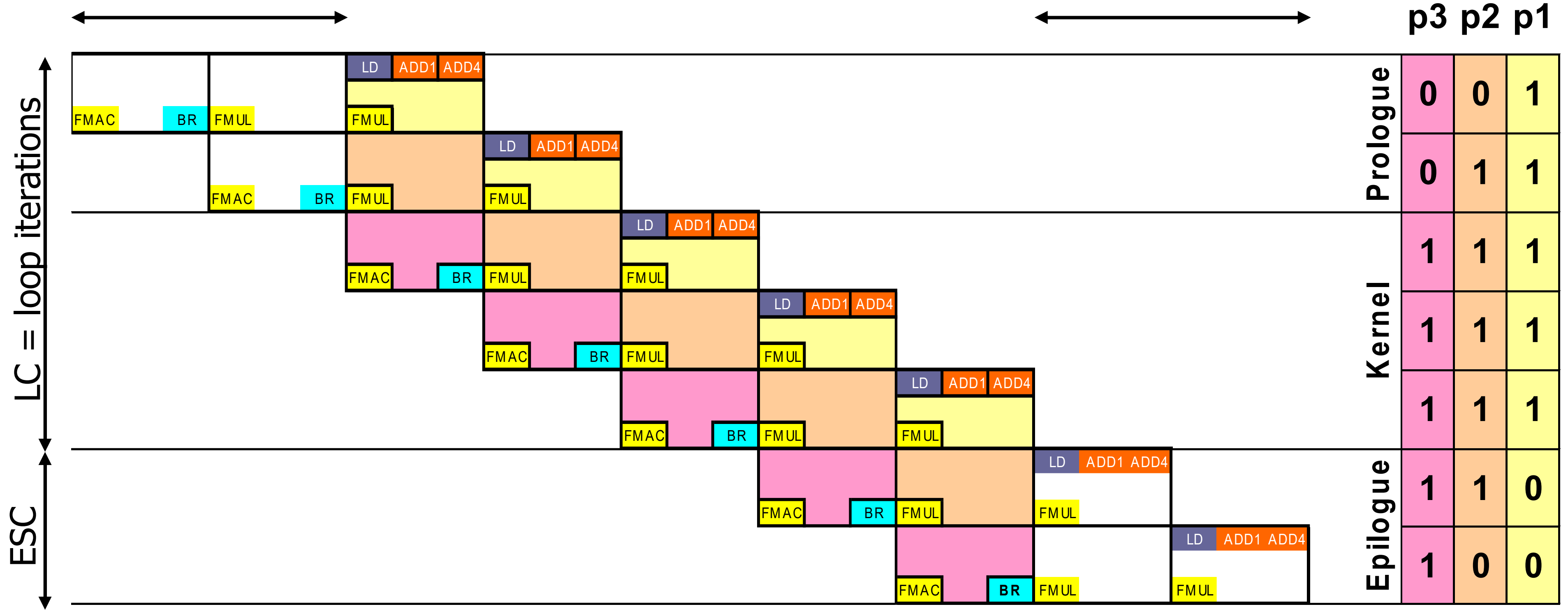
Software Pipelining



Modulo Scheduling

Due to predicate values, these instructions are discarded

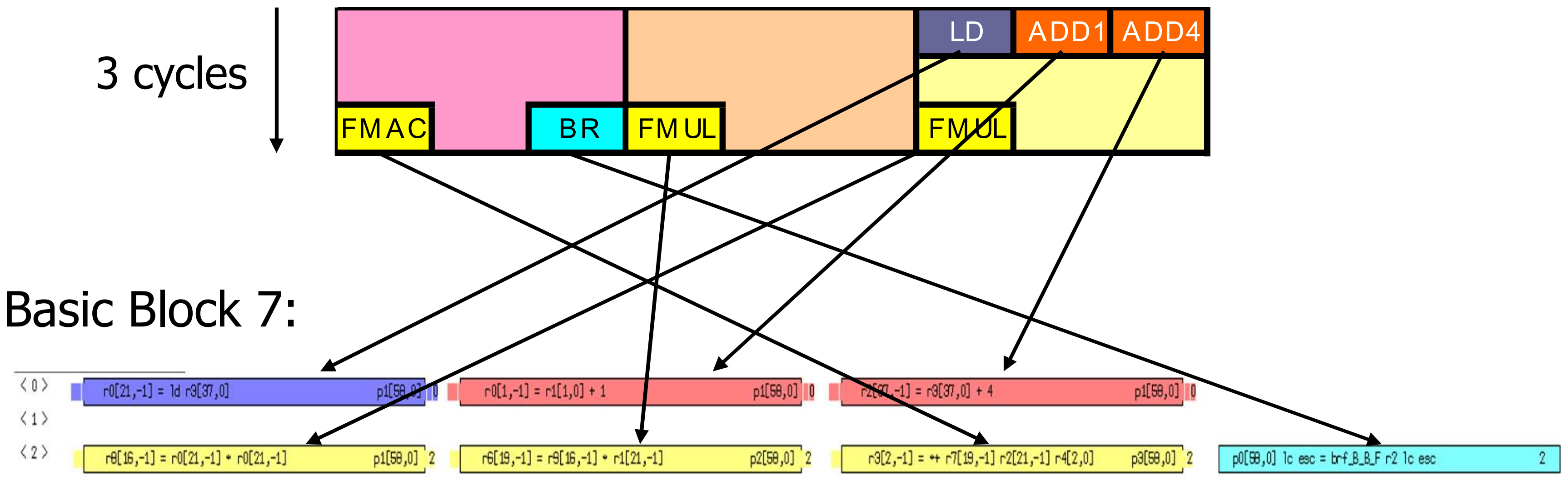
Due to predicate values, these instructions are discarded



	p3	p2	p1
Prologue	0	0	1
	0	1	1
Kernel	1	1	1
	1	1	1
	1	1	1
Epilogue	1	1	0
	1	0	0

Rotating Predicates

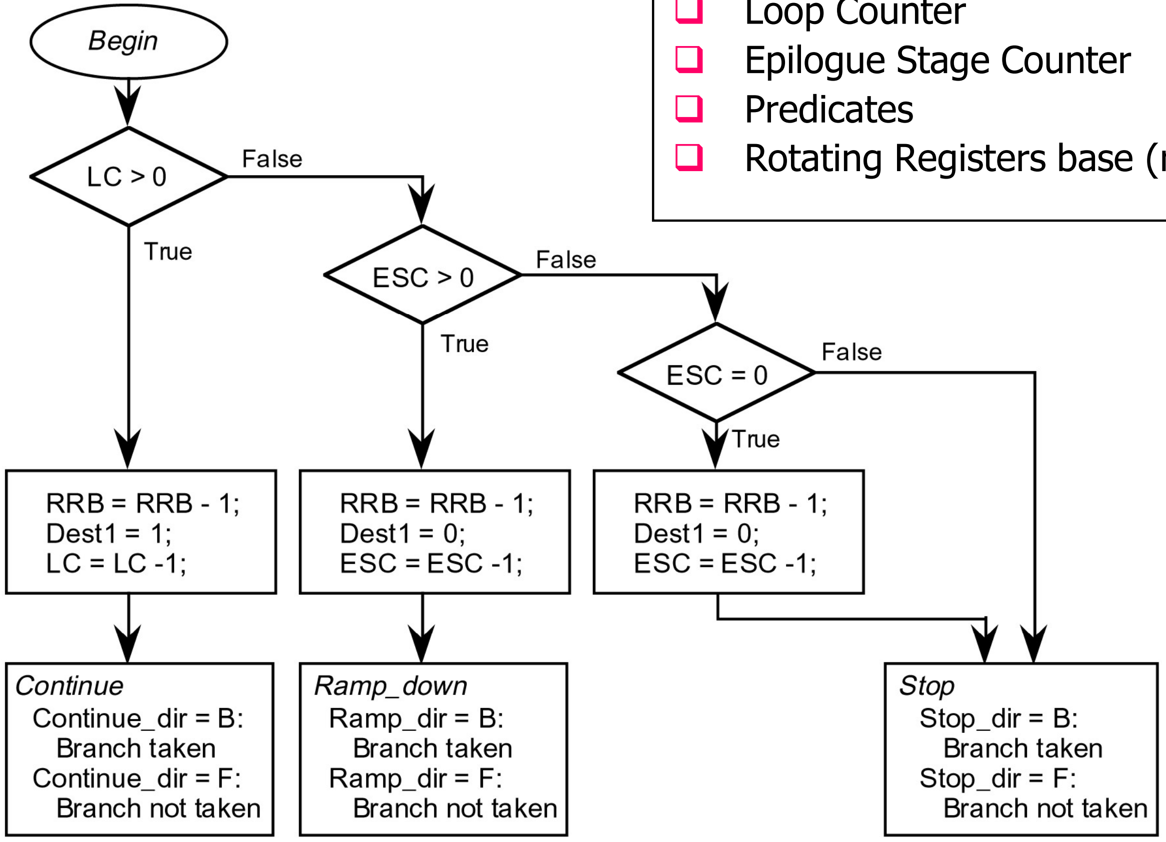
Scheduled Loop Kernel



```
r0[21,-1] = ld r3[37,0]            p1[58,0] 0
r0[1,-1] = r1[1,0] + 1             p1[58,0] 0
r2[37,-1] = r3[37,0] + 4           p1[58,0] 0
r8[16,-1] = r0[21,-1] * r0[21,-1]  p1[58,0] 2
r6[19,-1] = r9[16,-1] * r1[21,-1]  p2[58,0] 2
r3[2,-1] = ++ r7[19,-1] r2[21,-1] r4[2,0]  p3[58,0] 2
p0[58,0] lc esc = brf_B_B_F r2 lc esc  2
```

BRF: Branch FOR

Special instruction for Modulo Scheduling. It modifies
- ❑ Loop Counter
- ❑ Epilogue Stage Counter
- ❑ Predicates
- ❑ Rotating Registers base (r0→r1, r1→r2, etc.)



Comments:
Src1 is a BTR; Dest1 is a predicate register.
Branch address = src1.address

Disable Modulo Scheduling

- Create new parameter-set that disables modulo scheduling
  - Open “Parameters” “std”
  - Under “elcor”, open “DRIVER_DEFAULTS”
  - Turn off modulo scheduling

```
do_modulo_scheduling = no
```

  - “Save Changes As” “std_no_modulo”

No Modulo Scheduling Compilation

- Compile modulo_scheduling benchmark
  - Open “Compile”
  - Benchmark “modulo_scheduling”
  - Parameter-Set “std_no_modulo”
  - Project-File “modulo_scheduling”
  - Compilation and Simulation
    - Basic Blocks
    - Code Generation
    - Processor Simulation
  - GO!

Modulo Scheduling Analysis

- Looking at the generated code in View IR, we notice that the loop body is 3 cycles, why is that?
- Compare the performance with and without modulo scheduling: number of cycles, size of code, and IPC, and explain the differences
- What's the main difference between modulo scheduling and loop unrolling

Resources modification

- Set the number of floating point units back to 2 (Slide 9)
- Will the length of the loop body change? Why?
- Verify using simulation