Extraction of Efficient Instruction Schedulers from Cycle-true Processor Models

Oliver Wahlen,
Manuel Hohenauer,
Rainer Leupers,
Gerd Ascheid,
Heinrich Meyr
RWTH Aachen

Gunnar Braun
CoWare, Inc.

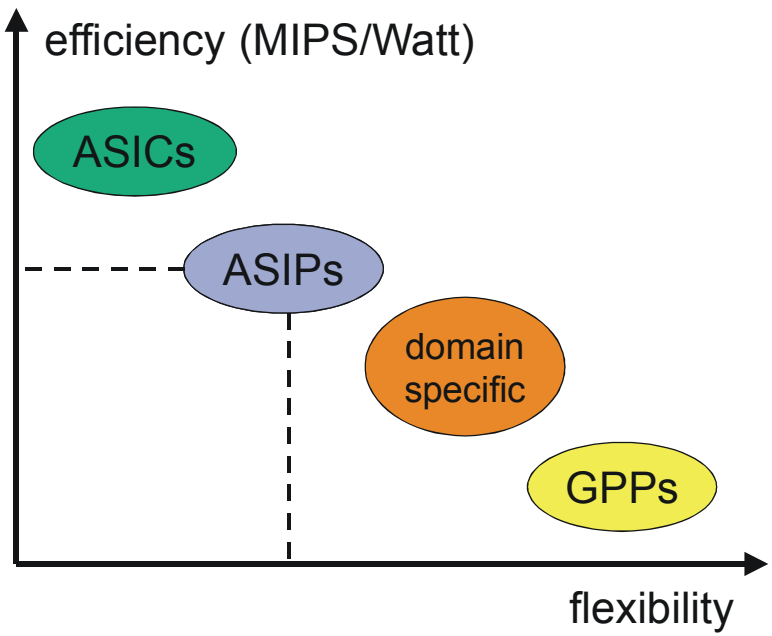
Xiaoning Nie
Infineon Technologies

Motivation: Why ASIPs?

Application Specific Instruction-Set Processors

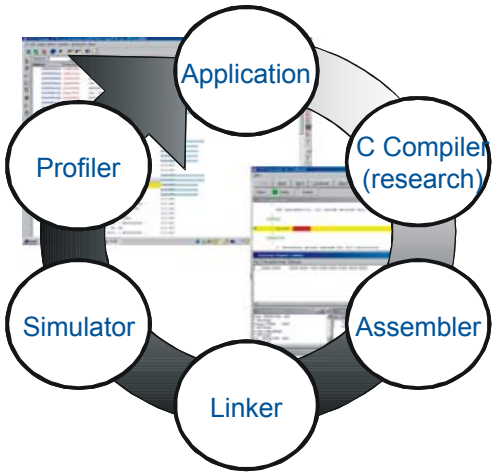
Combine advantages of processors and ASICs:

- **Provide system programmability and reconfigurability**
- **Good tradeoff: performance/power consumption/area**
- **Can easily be integrated into embedded systems**

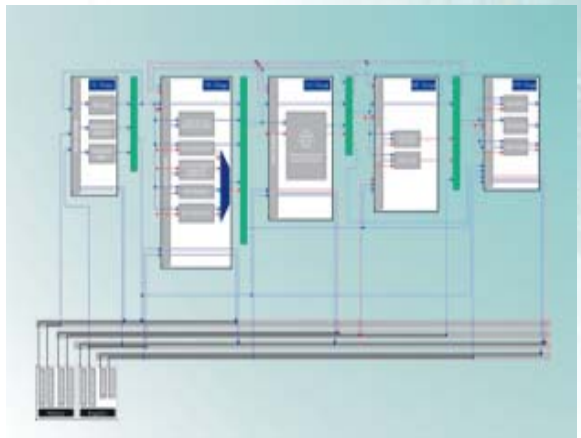


Solution: LISA Processor Design Platform

Language for Instruction-Set Architectures



EDGE™ Processor Designer

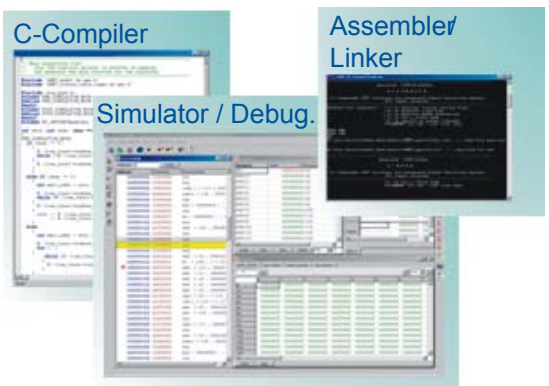


Architecture Exploration

LISA 2.0
Architecture
Specification

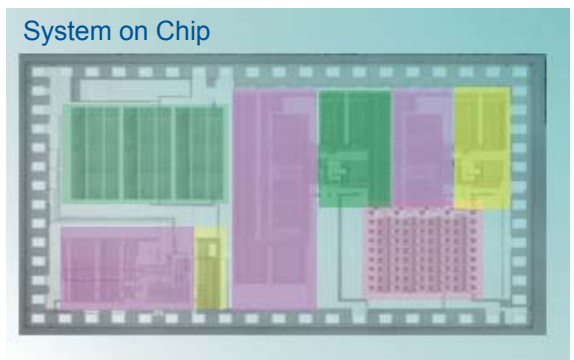
Architecture Implementation

RIM™ Software Designer



Software Application Design

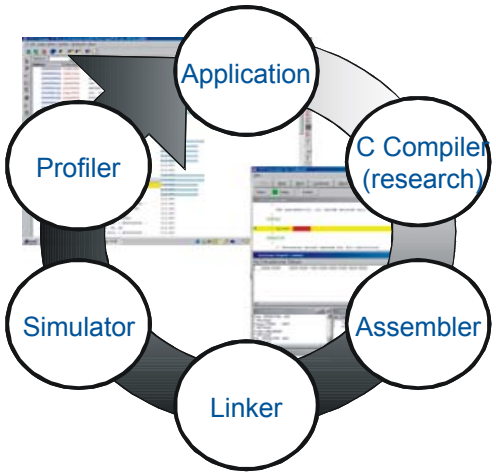
HUB™ System Integrator



Integration and Verification

Solution: LISA Processor Design Platform

Language for Instruction-Set Architectures



Architecture Exploration

EDGE™
Designe

LIS
Arch

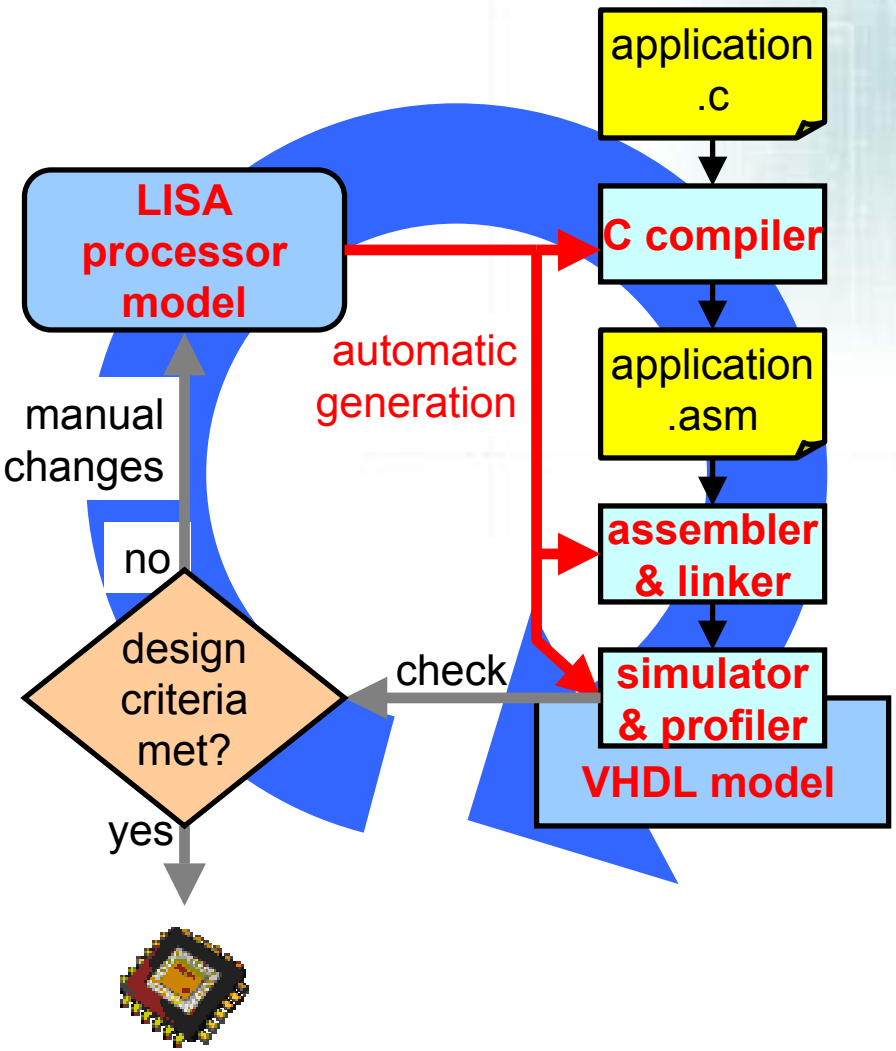
Architecture Exploration Loop

Automatic tool generation:

- Speeds up design cycles
- Eliminates consistency problem

C – compiler in the loop:

- Reduction in implementation and verification time
- IP reuse



Compiler Structure and Generation

LISA processor model

Semiautomatic Generation

.cgd

compiler backend description

Generation

CoSy Compiler Development System

.c

C front-end engine

IR

optimizations

Instruction Selector

Register Allocator

Scheduler

Emitter

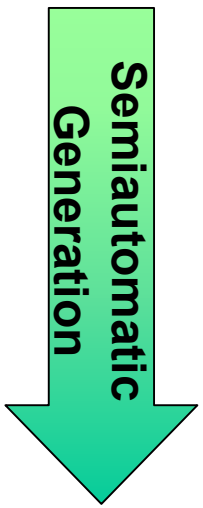
architecture specific backend engines

.asm

Scheduler Generation

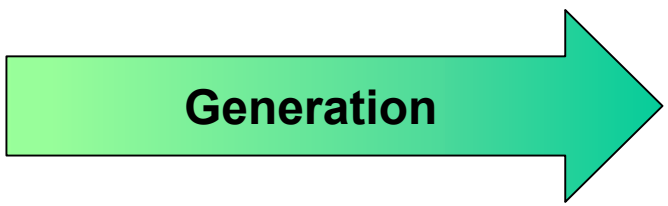
LISA processor model

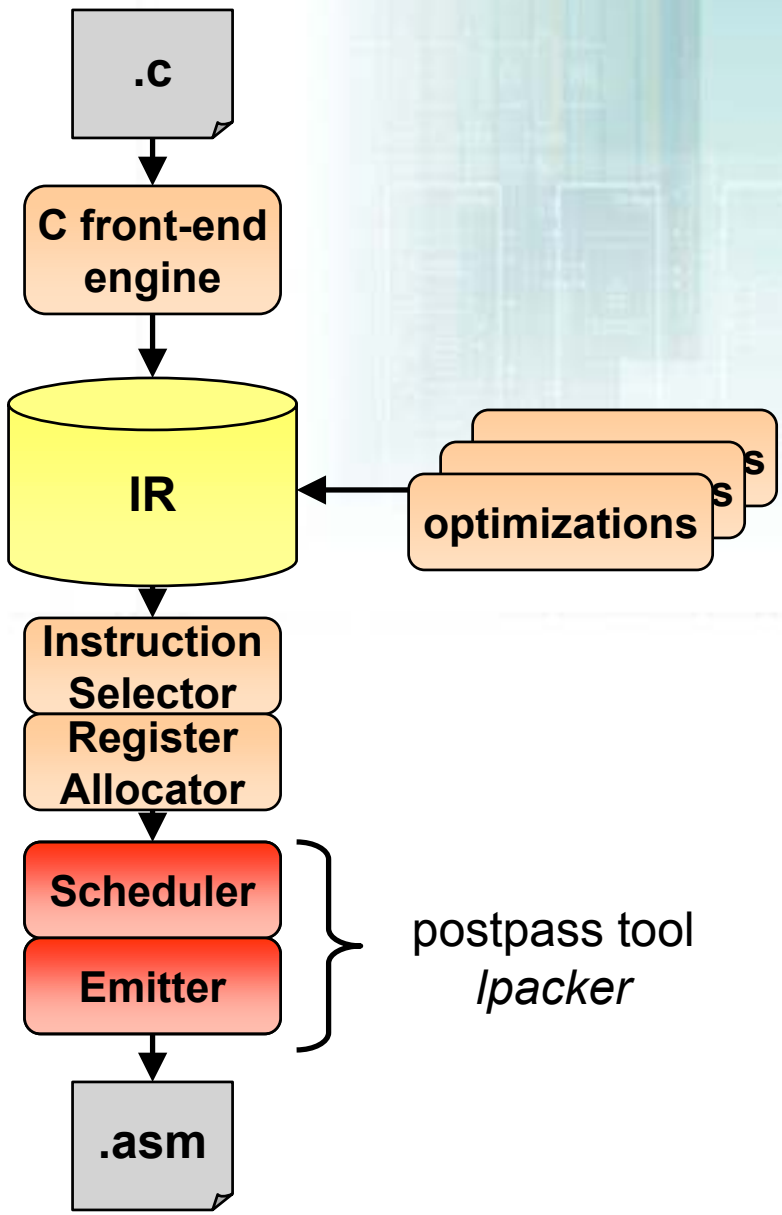
```
[EXPRESSION,
 PEAS-III]
```



.cgd

compiler backend description





CoSy Compiler Development System

Scheduler Description

Reservation Tables

[O.Wahlen, M.Hohenauer, R.Leupers, H. Meyr, 2003]

	ALU_op	MUL_op
cycle 0		
cycle 1		
cycle 2	EX_alu	EX_mul
cycle 3		EX_mul
cycle 4		

Elimination of Structural Hazards

Example:

0: MUL R1,R2,R3
1: NOP
2: MUL R4,R5,R6

Scheduler Description

Latency Tables

RAW	ALU_in	MUL_in
ALU_out	1	1
MUL_out	2	2

WAW

WAR

Elimination of Dataflow Hazards

Example:

```
0: MUL R3,R1,R2
1: NOP
2: ADD R5,R3,R4
```

Reservation Tables

[O.Wahlen, M.Hohenauer, R.Leupers, H. Meyr, 2003]

	ALU_op	MUL_op
cycle 0		
cycle 1		
cycle 2	EX_alu	EX_mul
cycle 3		EX_mul
cycle 4		

Elimination of Structural Hazards

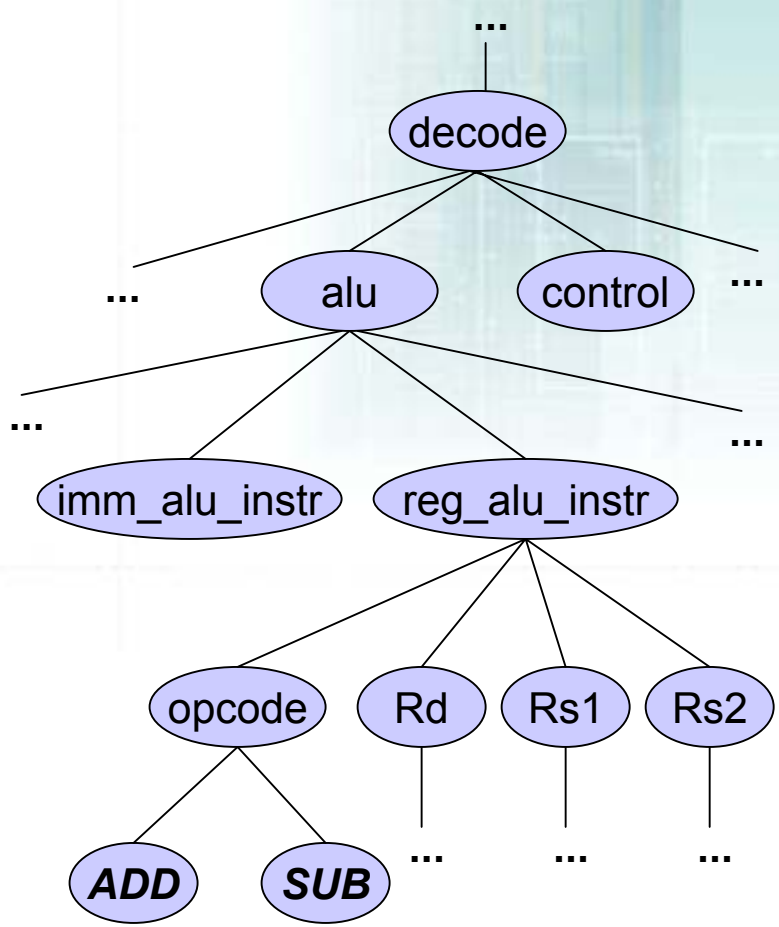
Example:

```
0: MUL R1,R2,R3
1: NOP
2: MUL R4,R5,R6
```

LISA Description

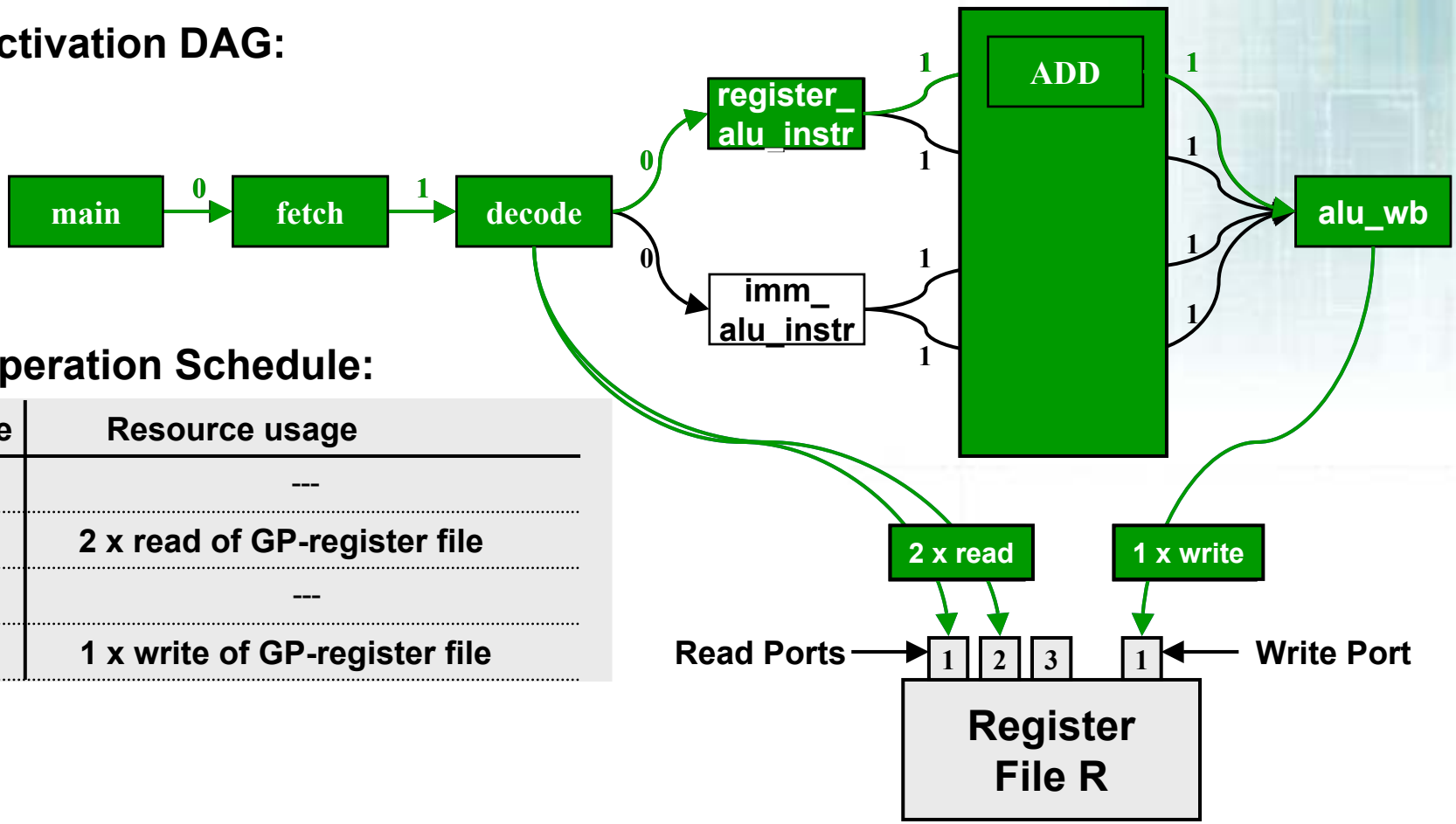
```
OPERATION reg_alu_instr IN pipe.ID
{
  DECLARE {
    GROUP Opcode = { ADD || SUB };
    GROUP Rs1, Rs2, Rd = { gp_reg };
  }
  CODING { Opcode Rs2 Rs1 Rd 0b0[10] }
  SYNTAX { Opcode ~" " Rd ~" " Rs1 ~" " Rs2 }

  BEHAVIOR {
    PIPELINE_REGISTER(pipe,ID/EX).src1 = GP_Regs[Rs1];
    PIPELINE_REGISTER(pipe,ID/EX).src2 = GP_Regs[Rs2];
    PIPELINE_REGISTER(pipe,ID/EX).dst = Rd;
  }
  ACTIVATION { Opcode }
}
```



Scheduler Generation: Operation Schedule

Activation DAG:



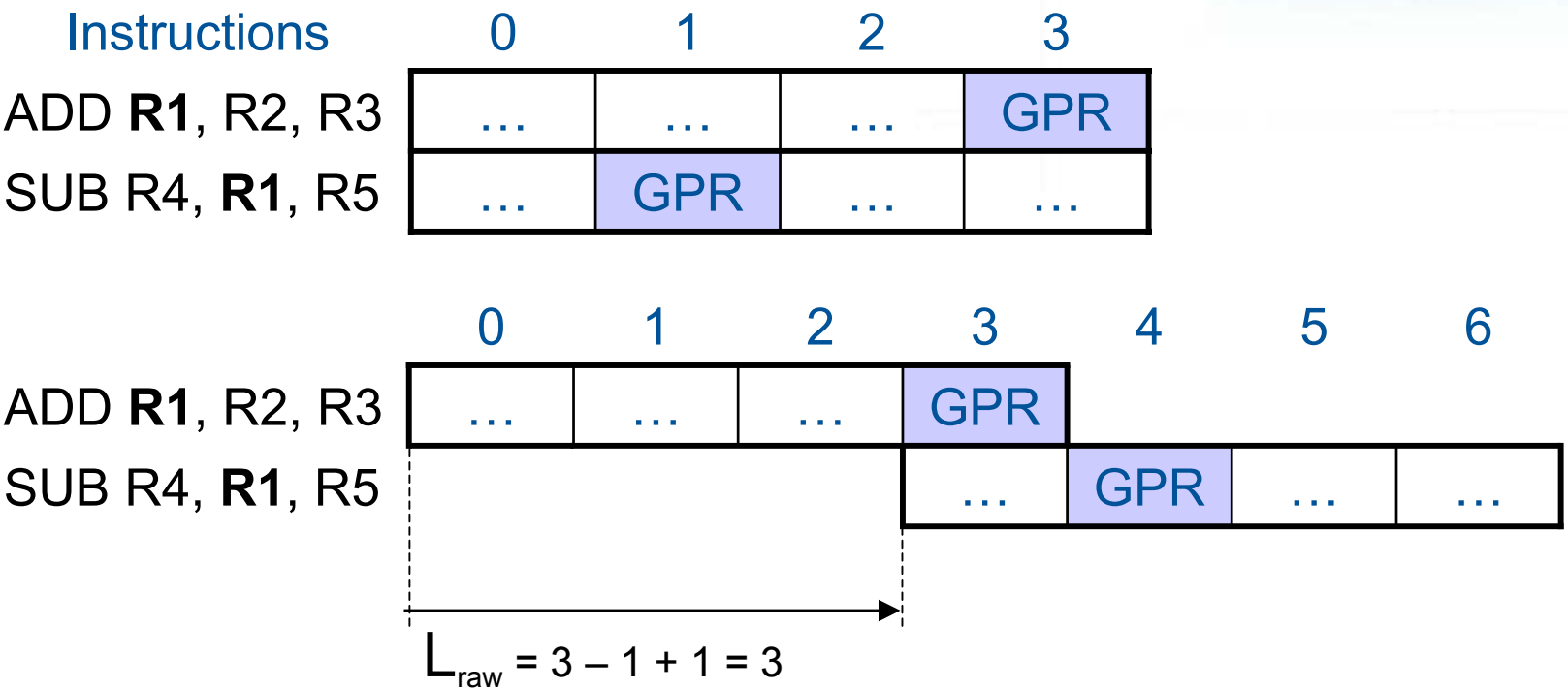
Operation Schedule:

Cycle	Resource usage
0	---
1	2 x read of GP-register file
2	---
3	1 x write of GP-register file

Latency Calculation

Latencies between two instructions *i* and *j* (*R* is a processor resource)

$$L_{raw}(i, j) = Max_R(\text{last\_write\_cycle}(j, R) - \text{first\_read\_cycle}(i, R) + 1)$$

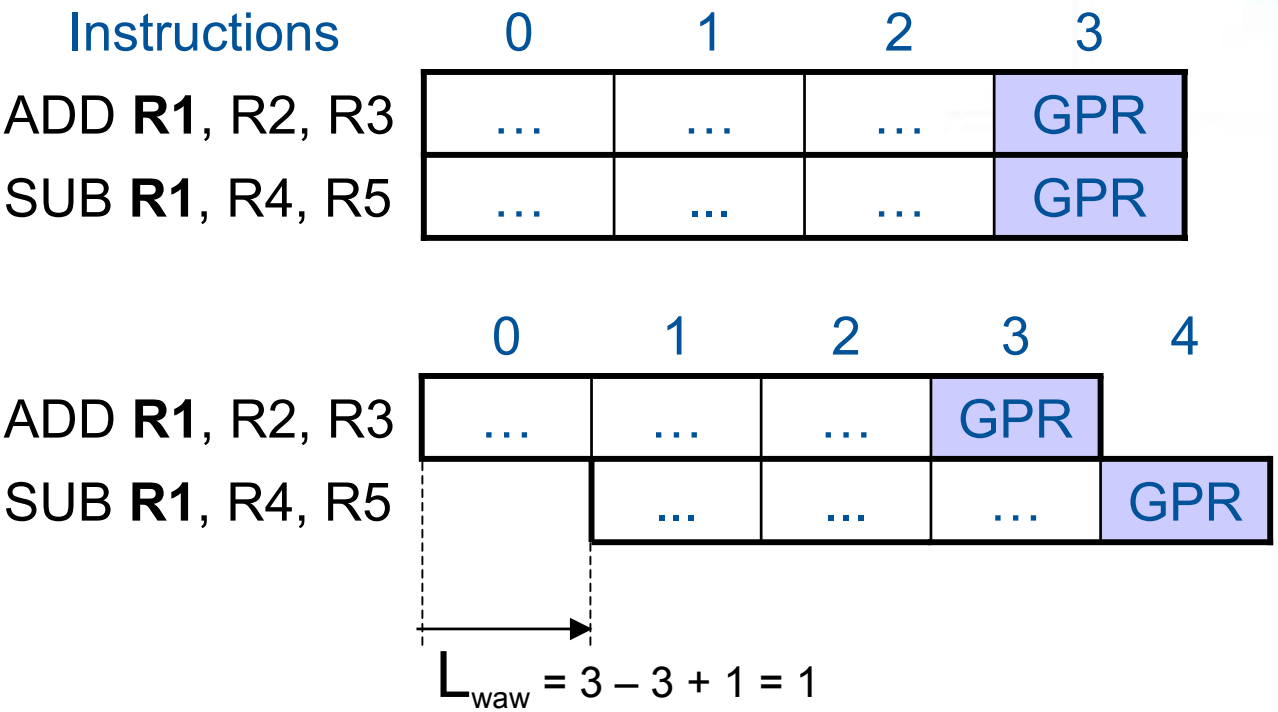


Latency Calculation

Latencies between two instructions *i* and *j* (*R* is a processor resource)

$$L_{raw}(i, j) = Max_R(\text{last\_write\_cycle}(j, R) - \text{first\_read\_cycle}(i, R) + 1)$$

$$L_{waw}(i, j) = Max_R(\text{last\_write\_cycle}(j, R) - \text{first\_write\_cycle}(i, R) + 1)$$



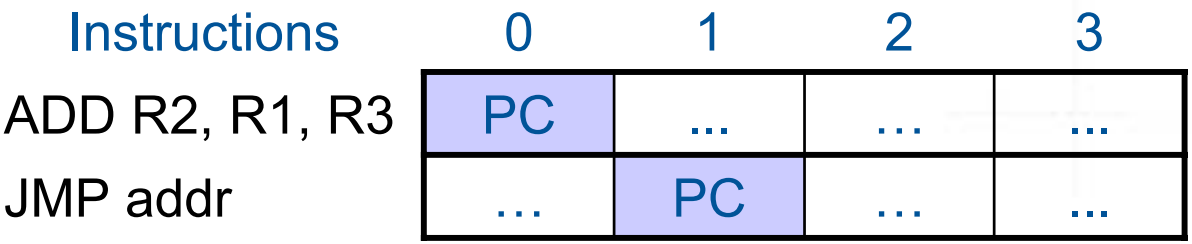
Latency Calculation

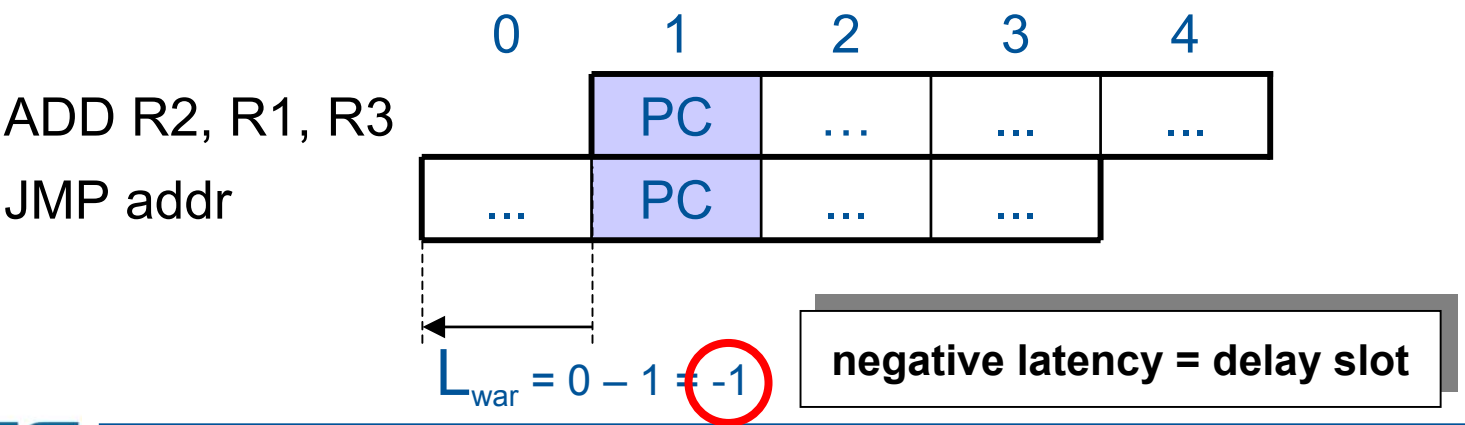
Latencies between two instructions *i* and *j* (*R* is a processor resource)

$L_{raw}(i, j) = Max_R(\text{last\_write\_cycle}(j, R) - \text{first\_read\_cycle}(i, R) + 1)$

$L_{waw}(i, j) = Max_R(\text{last\_write\_cycle}(j, R) - \text{first\_write\_cycle}(i, R) + 1)$

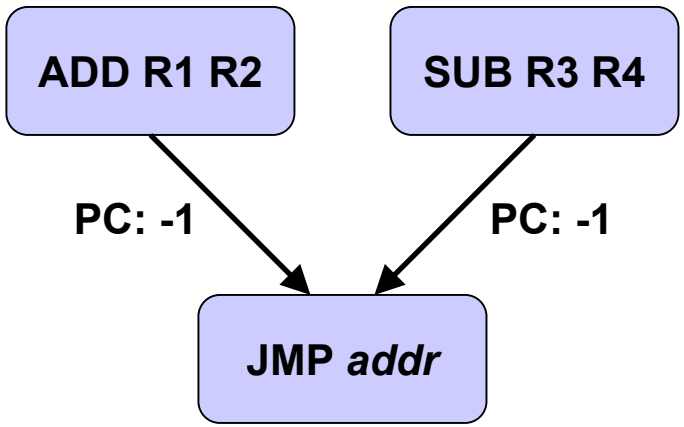
$L_{war}(i, j) = Max_R(\text{last\_read\_cycle}(j, R) - \text{first\_write\_cycle}(i, R))$



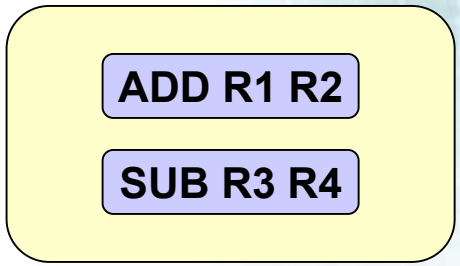


List Scheduling Example

data dependence dag

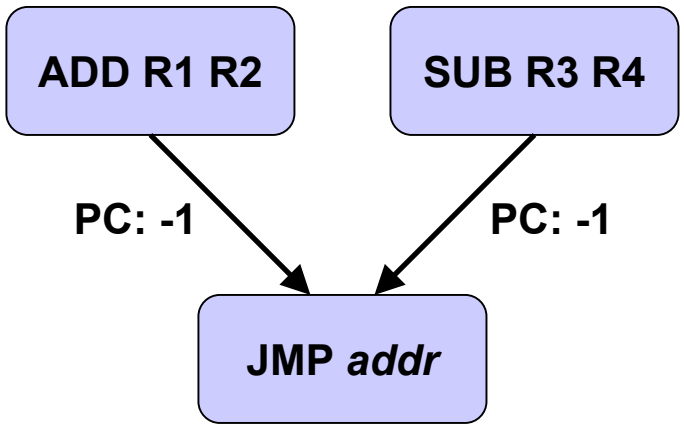


ready set:

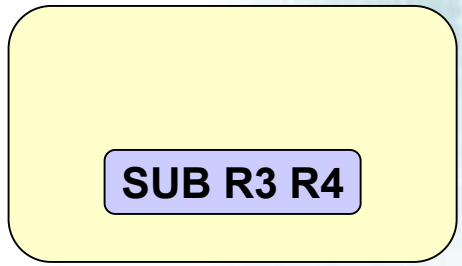


List Scheduling Example

data dependence dag



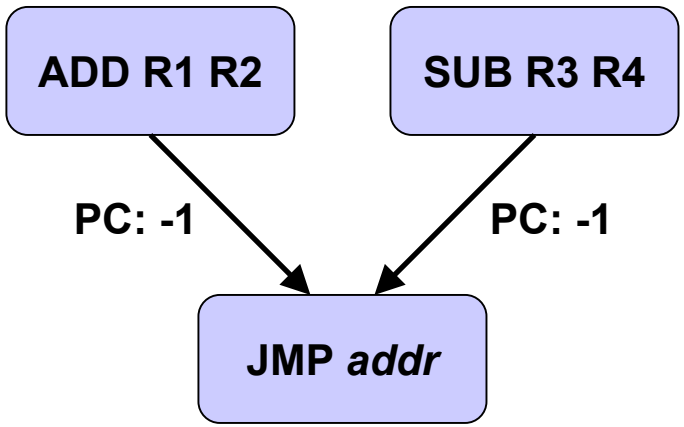
ready set:



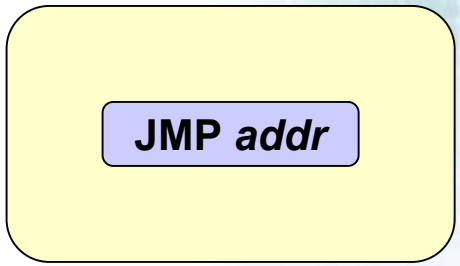
Cycle	Step 1
0	ADD R1 R2
1	
2	
3	

List Scheduling Example

data dependence dag



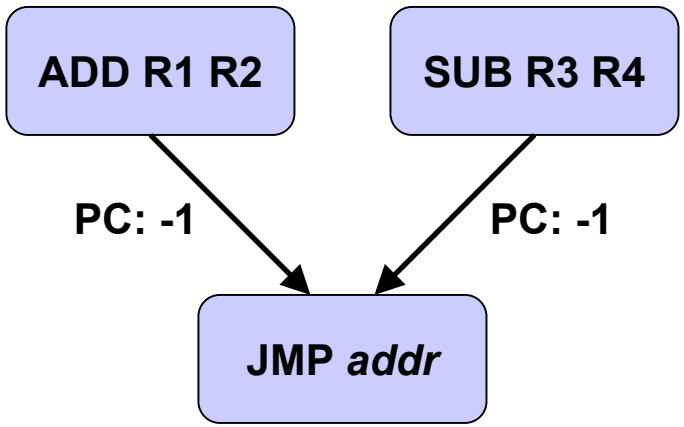
ready set:



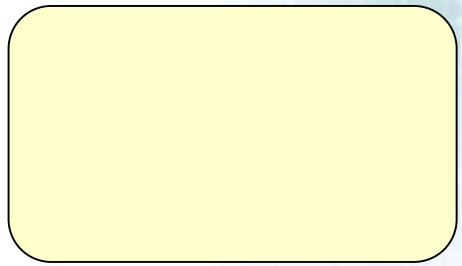
Cycle	Step 1	Step 2
0	ADD R1 R2	ADD R1 R2
1		SUB R3 R4
2		
3		

List Scheduling Example

data dependence dag



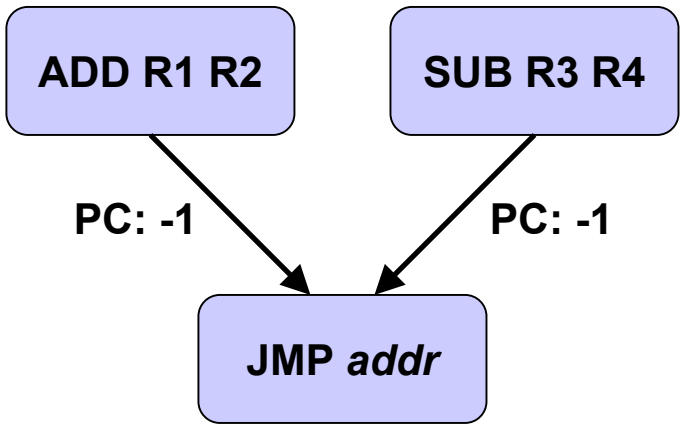
ready set:



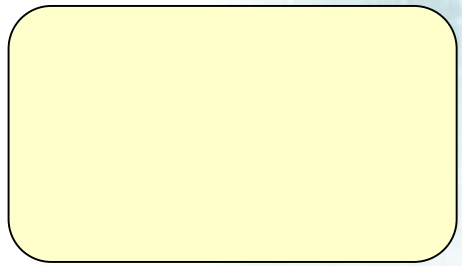
Cycle	Step 1	Step 2	Step 3
0	ADD R1 R2	ADD R1 R2	ADD R1 R2
1		SUB R3 R4	SUB R3 R4
2			JMP *addr*
3			

List Scheduling Example

data dependence dag



ready set:



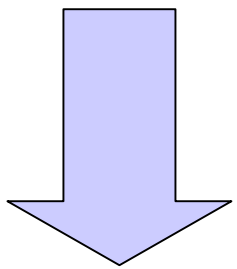
delay slot must be filled

Cycle	Step 1	Step 2	Step 3	Step 4
0	ADD R1 R2	ADD R1 R2	ADD R1 R2	ADD R1 R2
1		SUB R3 R4	SUB R3 R4	SUB R3 R4
2			JMP *addr*	JMP *addr*
3				NOP

Backtracking Scheduler

- Negative latencies can automatically be extracted from the LISA model
- They indicate delay slots

- Negative weights in dependence DAG cannot be utilized by list schedulers because scheduling decisions need to be revoked



Development of a retargetable Backtracking Scheduler

[S. G. Abraham, W. Meleis, I. D. Baev, 2000]

mixedBT Backtracking Scheduler

Concept: three scheduling modes

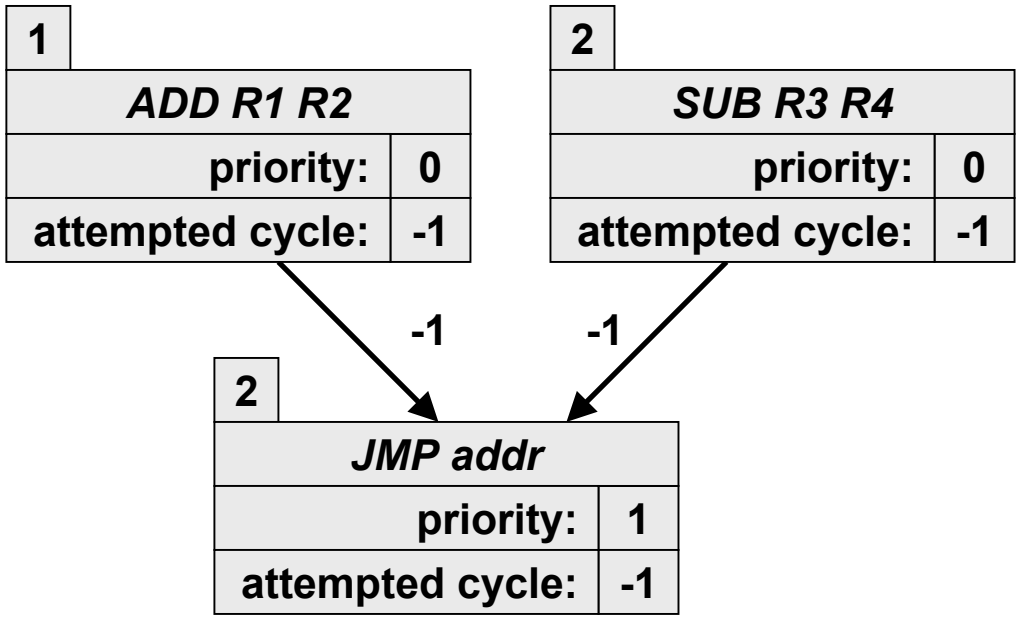
1. **normal scheduling:** if there is no conflict instructions are scheduled according to their data dependencies

2. **displace scheduling:** unschedule instructions that have lower priority and are causing a structural hazard

3. **force scheduling:** if 1 and 2 are not possible unschedule conflicts and force the scheduling of the candidate

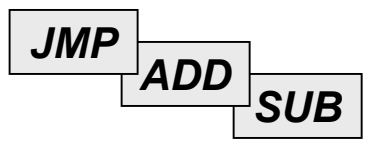
mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

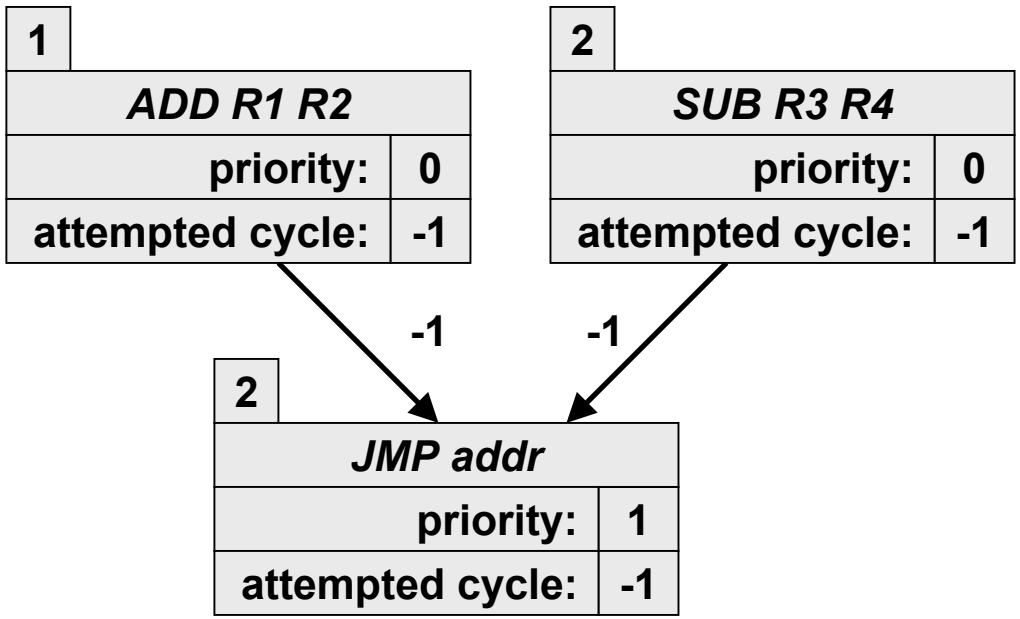
unscheduled list



Cycle											
0											
1											
2											
3											

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

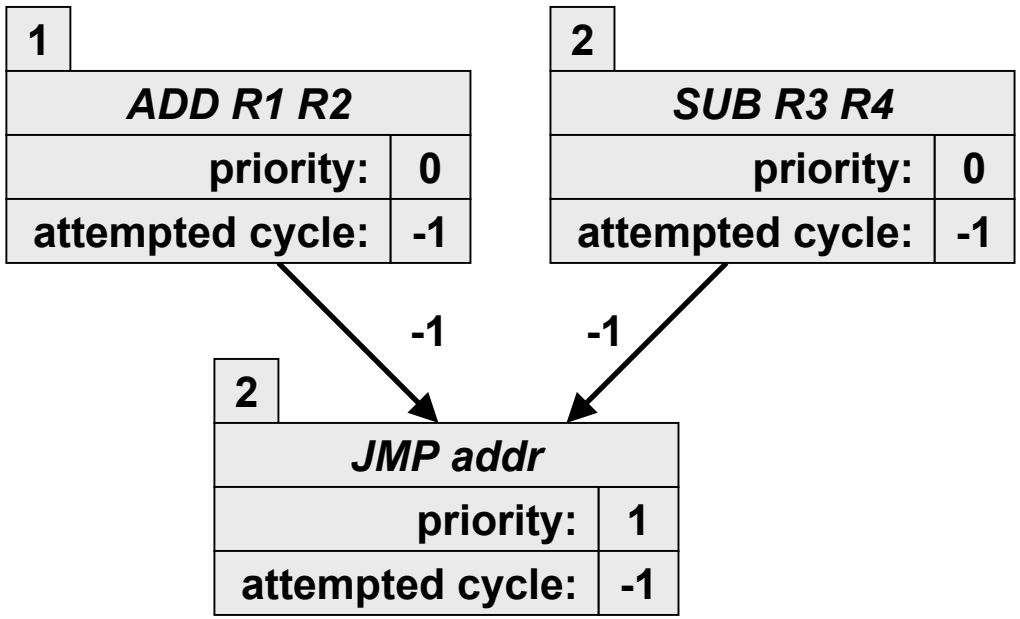
ADD *SUB*

Cycle	N										
0	*JMP*										
1											
2											
3											

N: normal, scheduled according to data dependencies

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

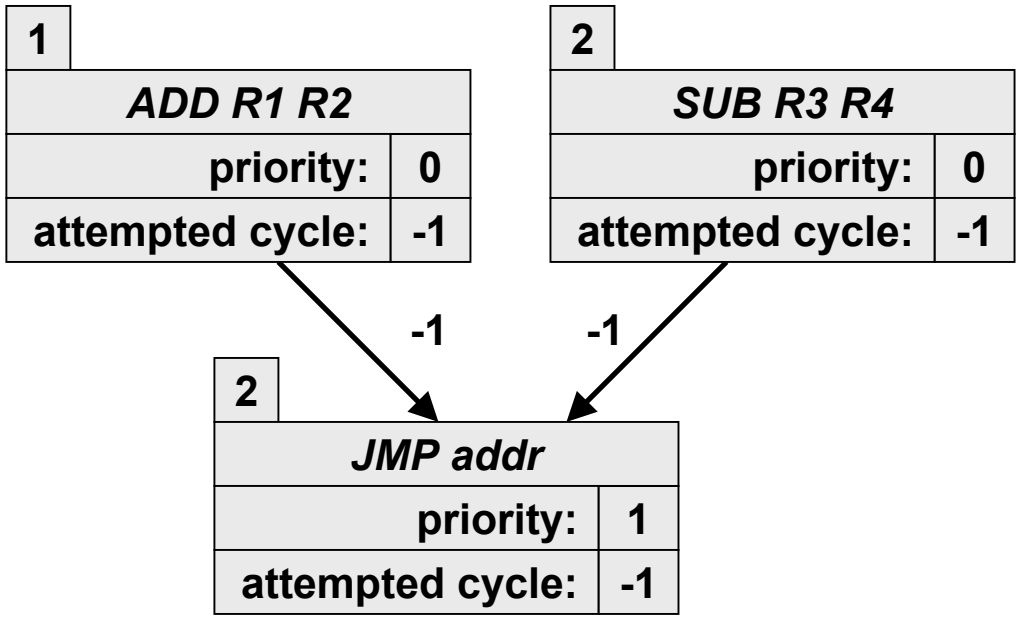
SUB

Cycle	N	N									
0	*JMP*	*JMP*									
1		*ADD*									
2											
3											

N: normal, scheduled according to data dependencies

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

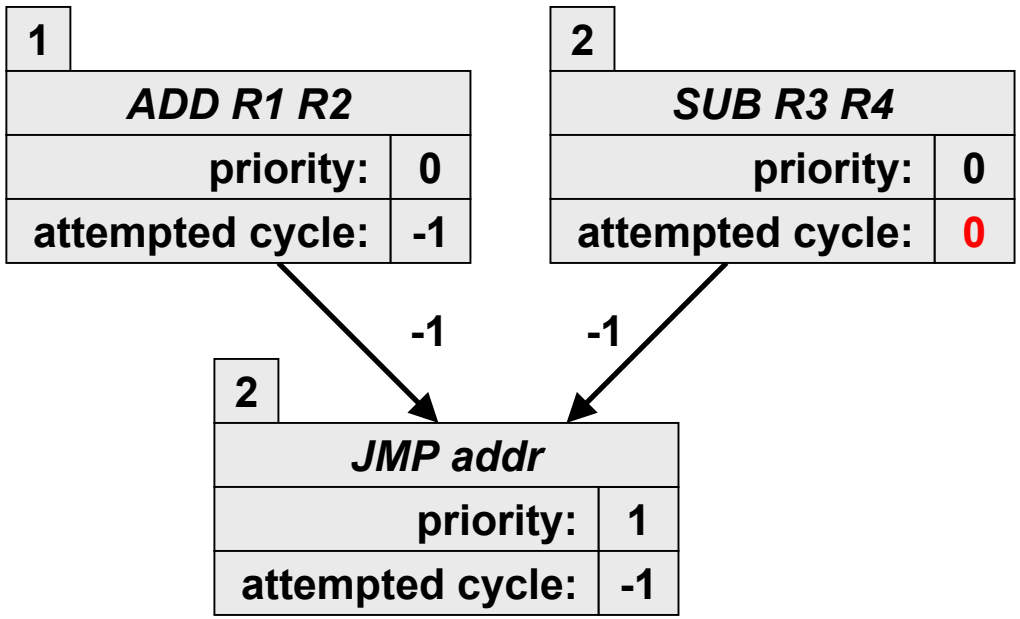
SUB

Cycle	N	N									
0	*JMP*	*JMP*	*MP*								
1		*ADD*	*ADD*								
2											
3											

normal and displace scheduling are not possible

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

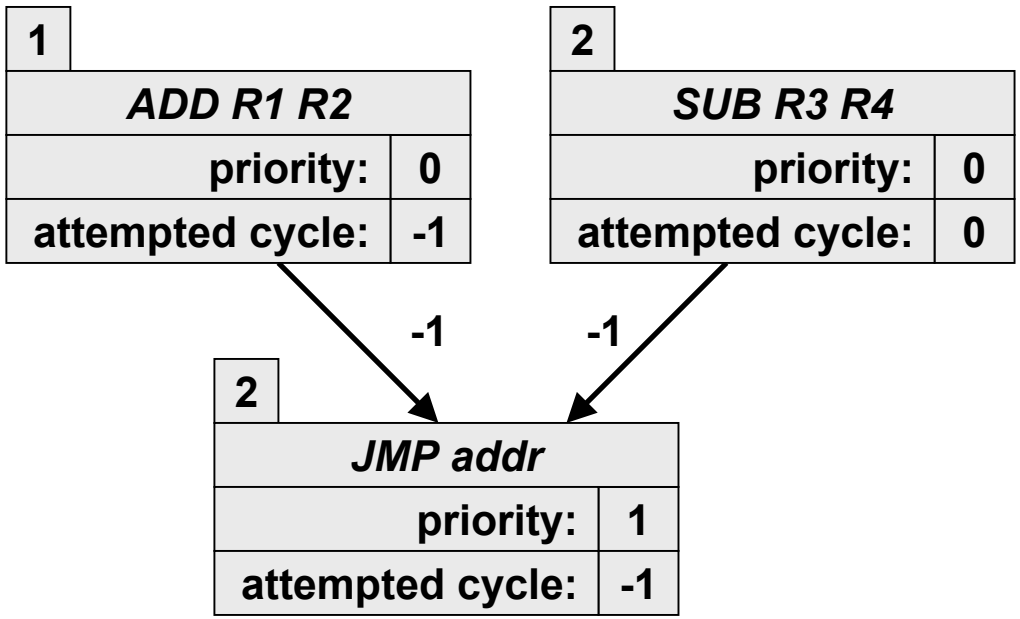
JMP

Cycle	N	N	F								
0	*JMP*	*JMP*	*SUB*								
1		*ADD*	*ADD*								
2											
3											

F: unschedule conflicts and force the scheduling

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

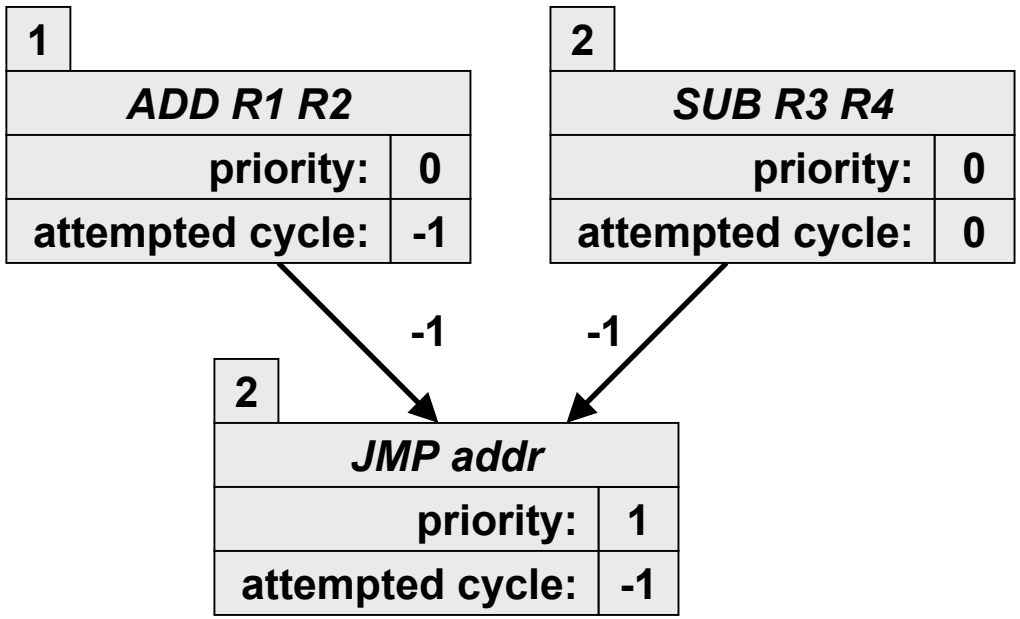
SUB

Cycle	N	N	F	D							
0	*JMP*	*JMP*	*SUB*	*JMP*							
1		*ADD*	*ADD*	*ADD*							
2											
3											

D: unschedule instructions that have lower priority & are causing a structural hazard

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

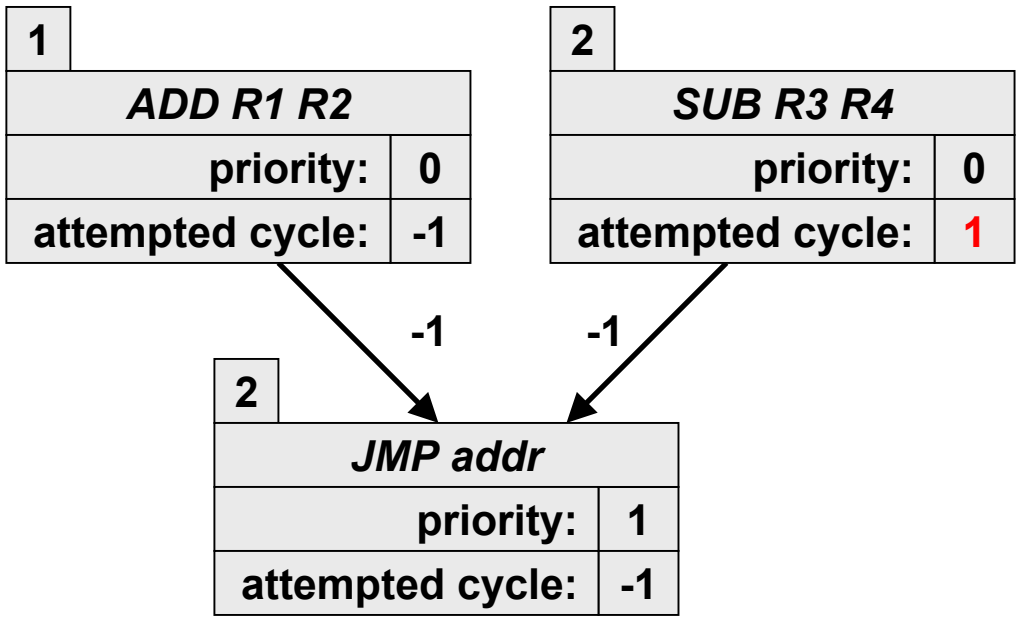
unscheduled list

					SUB						
Cycle	N	N	F	D							
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*						
1		*ADD*	*ADD*	*ADD*	*ADD*						
2											
3											

normal and displace scheduling are not possible

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

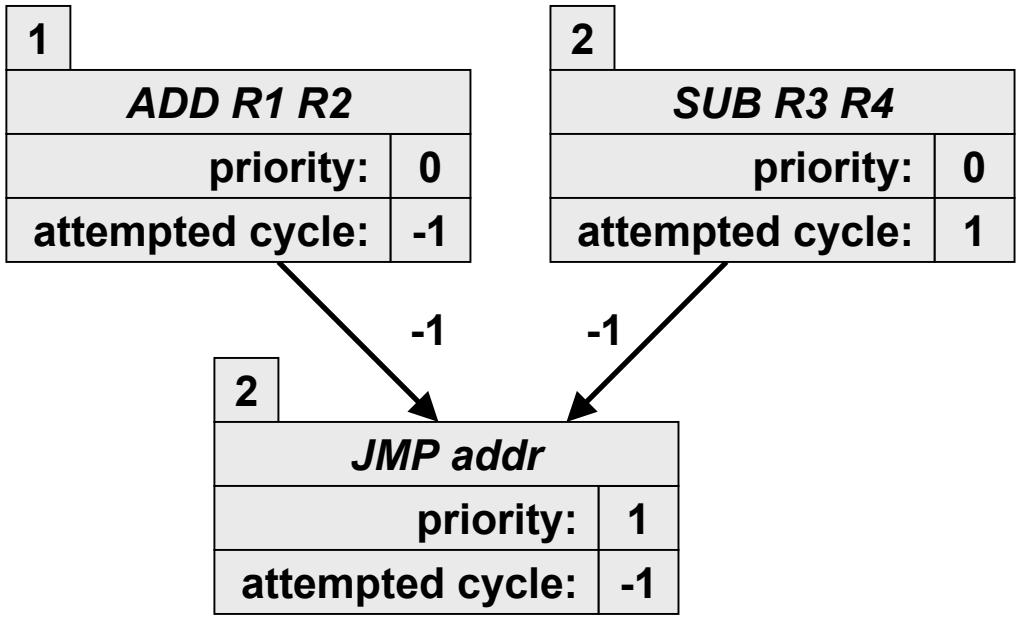
ADD

Cycle	N	N	F	D	F						
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*						
1		*ADD*	*ADD*	*ADD*	*SUB*						
2											
3											

F: unschedule conflicts and force the scheduling

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

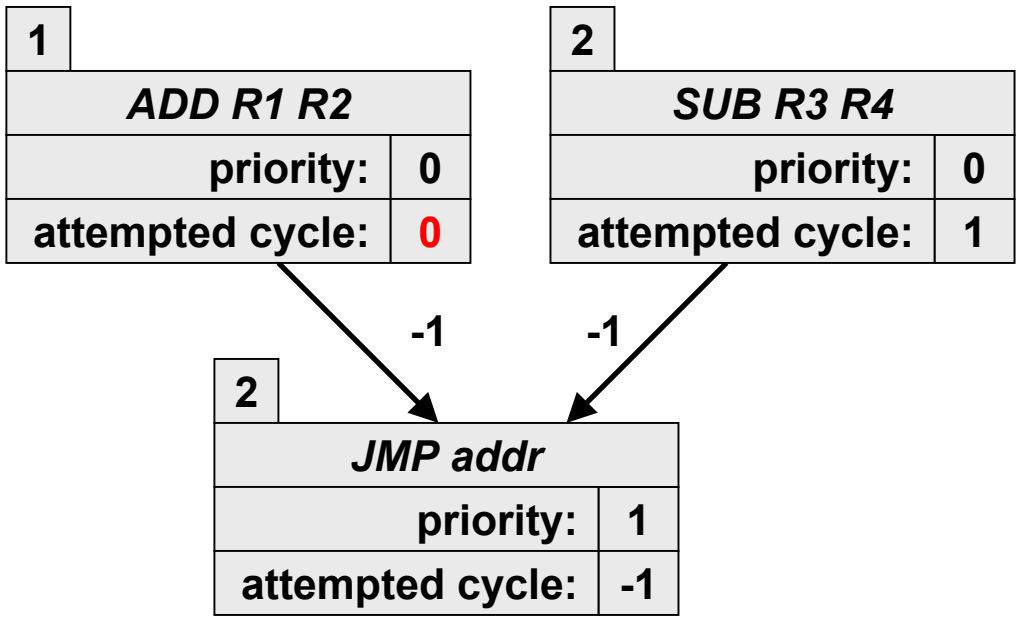
unscheduled list

						ADD					
Cycle	N	N	F	D	F						
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*JMP*					
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*					
2											
3											

normal and displace scheduling are not possible

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

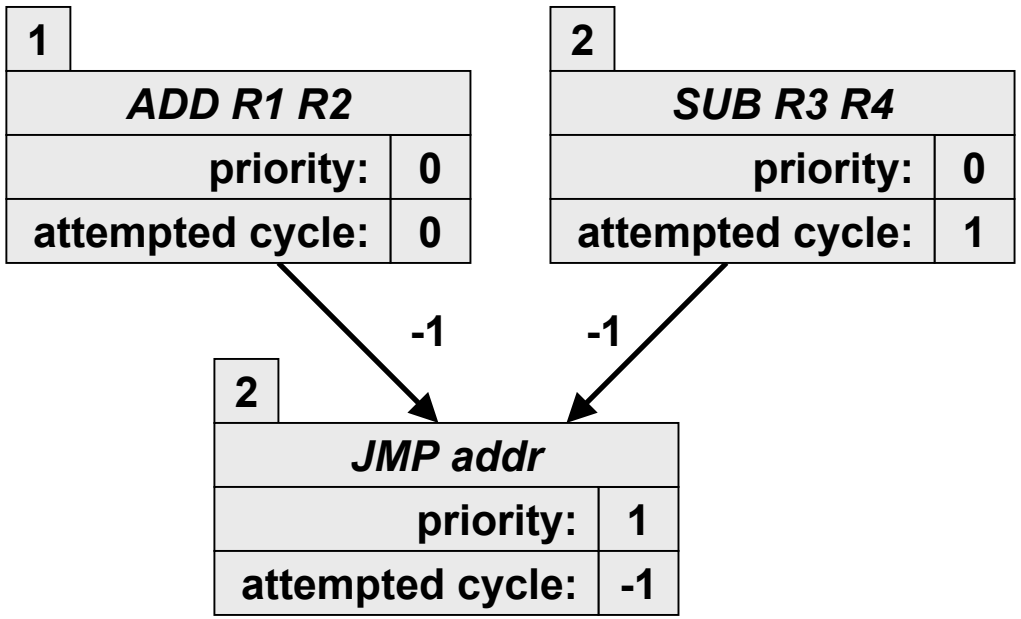
JMP

Cycle	N	N	F	D	F	F					
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*					
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*					
2											
3											

F: unschedule conflicts and force the scheduling

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

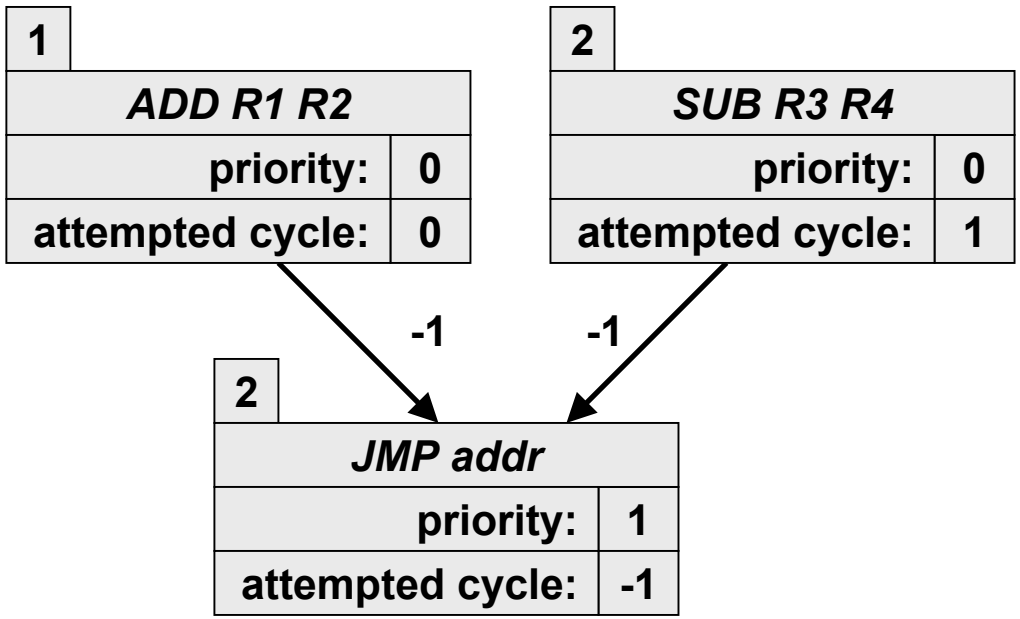
ADD

Cycle	N	N	F	D	F	F	D				
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*				
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*				
2											
3											

D: unschedule instructions that have lower priority & are causing a structural hazard

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

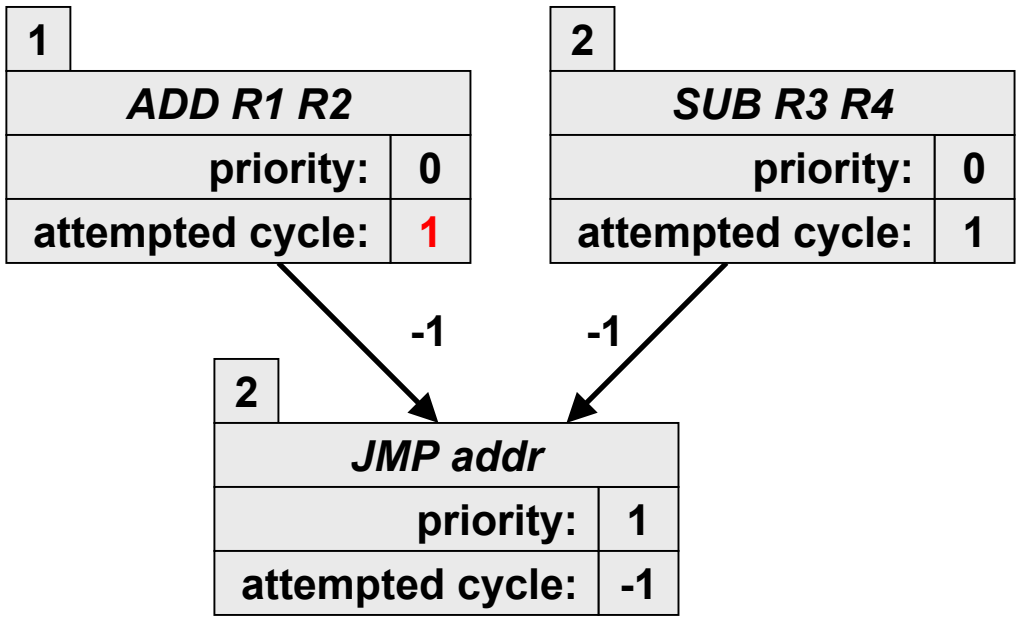
unscheduled list

								ADD			
Cycle	N	N	F	D	F	F	D				
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*	*JMP*			
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*	*SUB*			
2											
3											

normal and displace scheduling are not possible

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

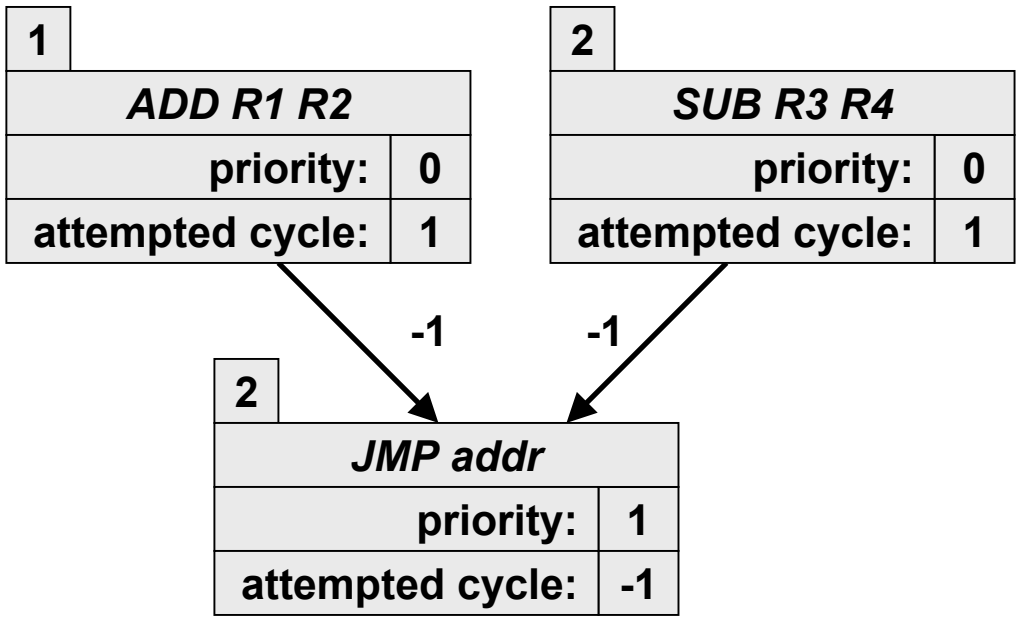
SUB

Cycle	N	N	F	D	F	F	D	F			
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*	*JMP*			
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*	*ADD*			
2											
3											

F: unschedule conflicts and force the scheduling

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

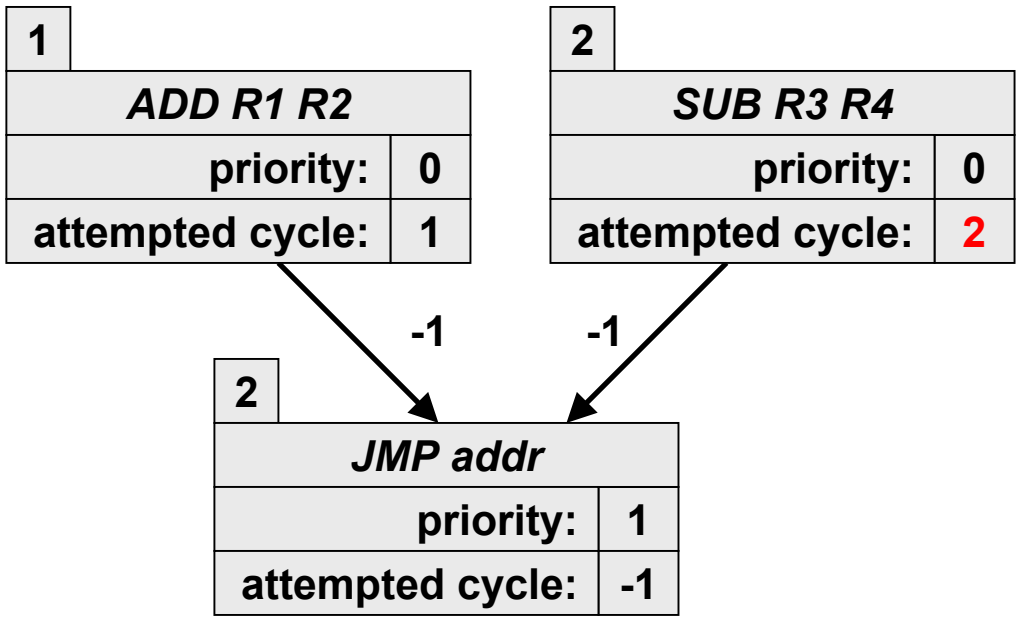
unscheduled list

Cycle	N	N	F	D	F	F	D	F	SUB		
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*	*JMP*	*JMP*		
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*	*ADD*	*ADD*		
2											
3											

normal and displace scheduling are not possible

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

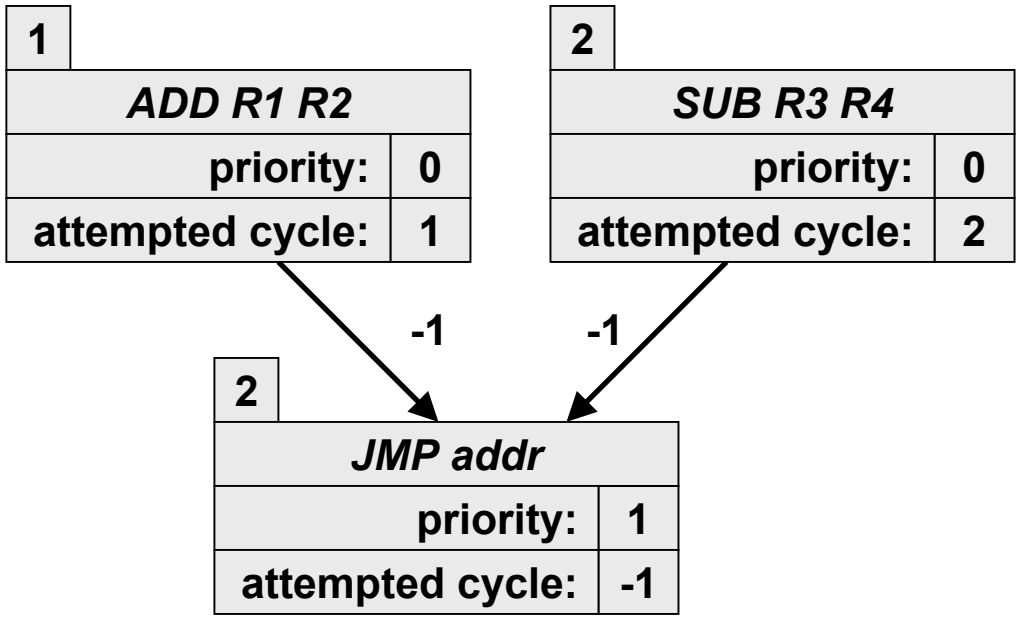
JMP

Cycle	N	N	F	D	F	F	D	F	F		
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*	*JMP*	*NOP*		
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*	*ADD*	*ADD*		
2									*SUB*		
3											

F: unschedule conflicts and force the scheduling

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

unscheduled list

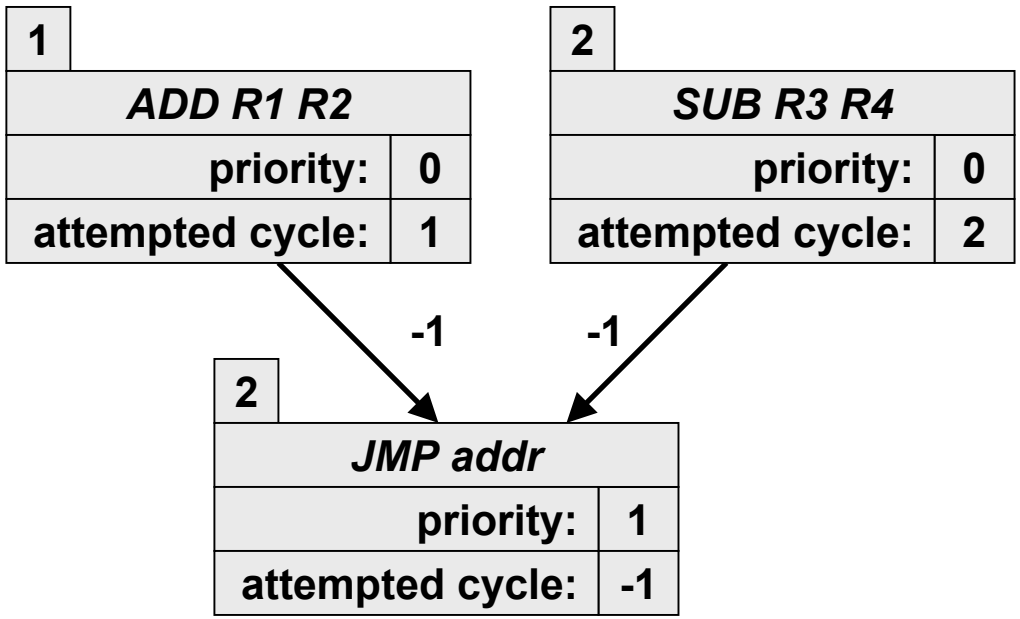
ADD

Cycle	N	N	F	D	F	F	D	F	F	D	
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*	*JMP*	*NOP*	*NOP*	
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*	*ADD*	*ADD*	*JMP*	
2									*SUB*	*SUB*	
3											

D: unschedule instructions that have lower priority & are causing a structural hazard

mixedBT Backtracking Scheduler Example

data dependence dag




```
initialize_priorities
loop(until_all_insn_scheduled)
{
  try normal_schedule
  or displace_schedule
  else
  {
    force_schedule
    update_attempted_cycle
  }
}
```

delay slot automatically filled

Cycle	N	N	F	D	F	F	D	F	F		N
0	*JMP*	*JMP*	*SUB*	*JMP*	*JMP*	*ADD*	*JMP*	*JMP*	*NOP*	*A*	*ADD*
1		*ADD*	*ADD*	*ADD*	*SUB*	*SUB*	*SUB*	*ADD*	*ADD*	*JMP*	*JMP*
2									*SUB*	*SUB*	*SUB*
3											

N: normal, scheduled according to data dependencies

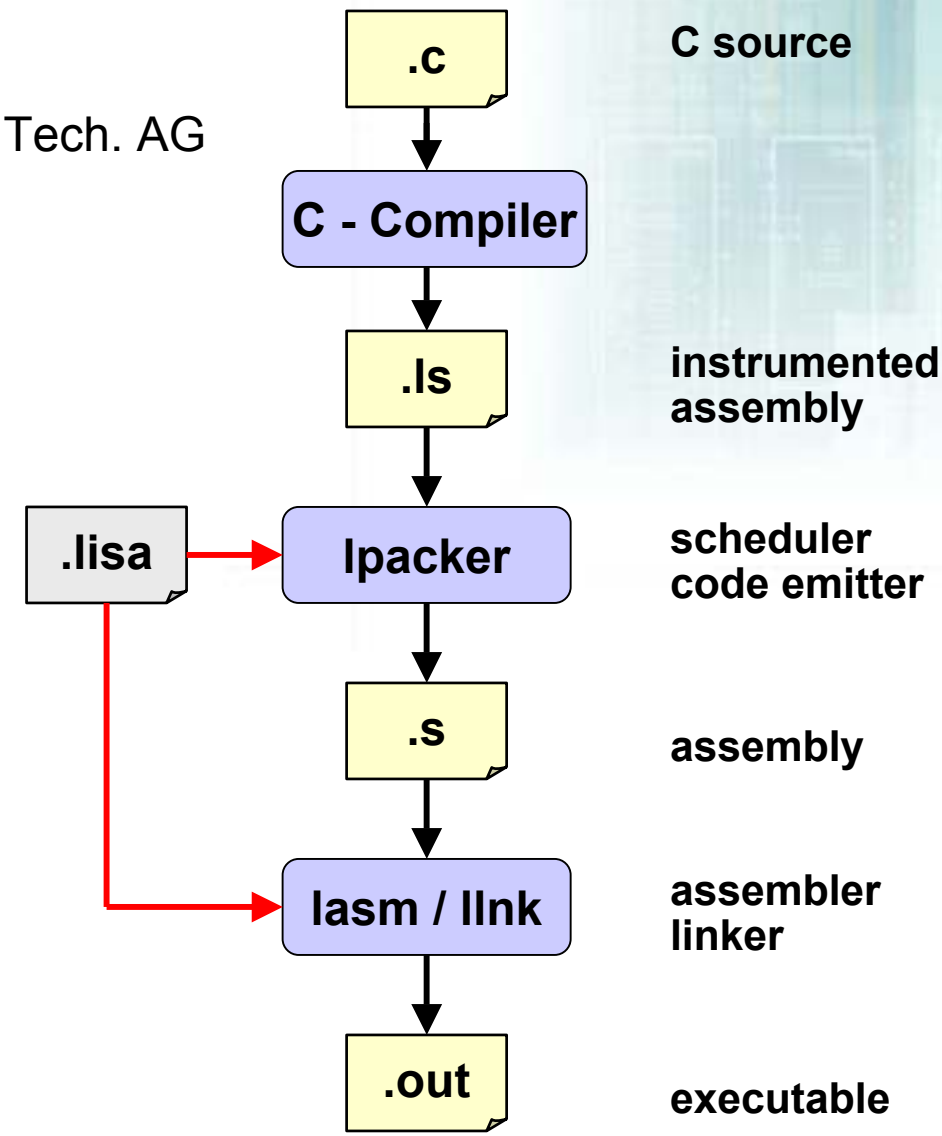
Toolflow

Target Architecture:

PP32 Network Processor, Infineon Tech. AG

Compilers:

- CoSy Compiler with LPacker as Scheduler/Emitter
- LCC (Little C Compiler, Princeton Univ.) with LPacker as Scheduler/Emitter



Toolflow

Target Architecture:

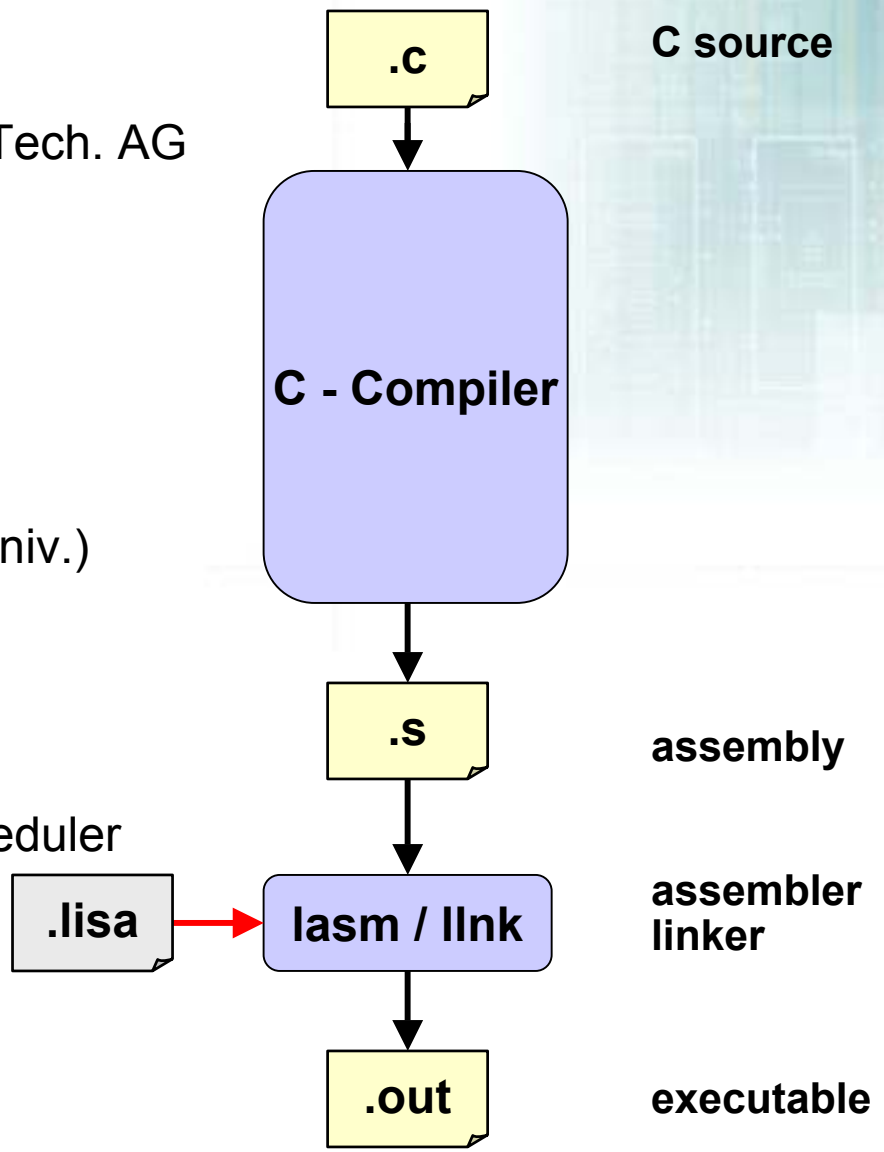
PP32 Network Processor, Infineon Tech. AG

Compilers:

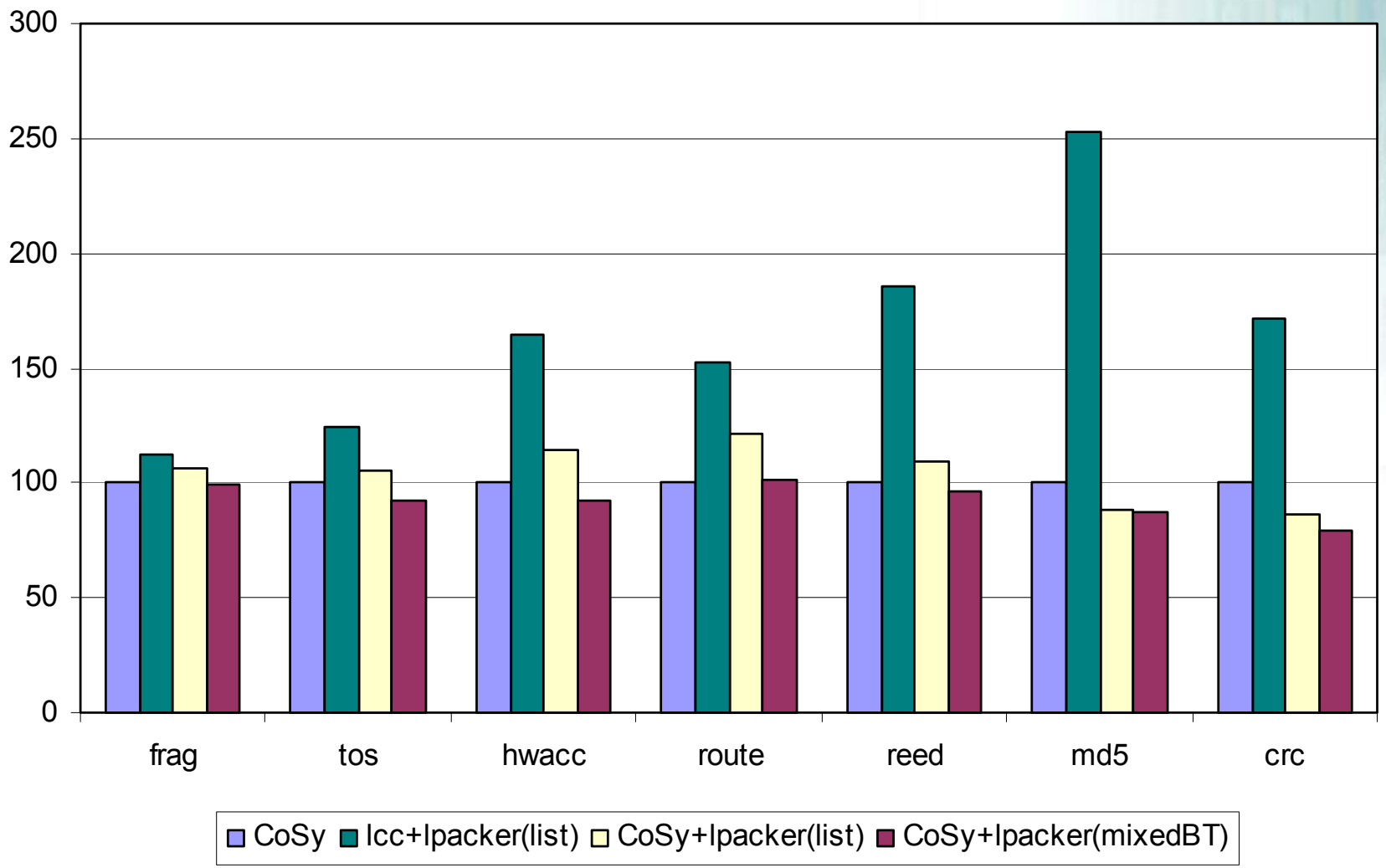
- CoSy Compiler with LPacker as Scheduler/Emitter
- LCC (Little C Compiler, Princeton Univ.) with LPacker as Scheduler/Emitter

Reference Compiler:

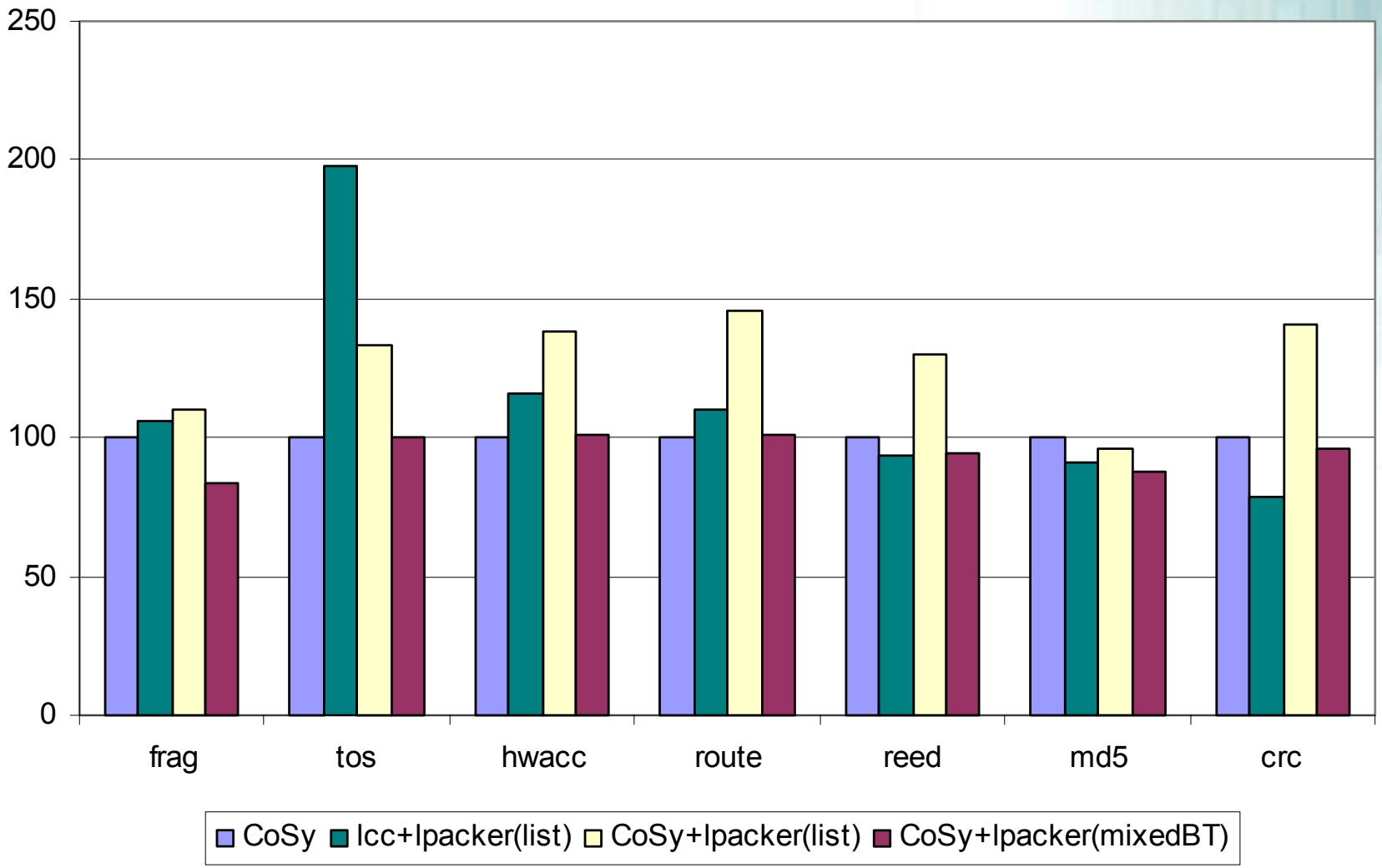
- CoSy Compiler with native List Scheduler



Results: Rel. Cycle Count for PP32 NPU



Results: Rel. Code Size for PP32 NPU



Summary

- It is possible to extract all scheduler related information from LISA processor models
- Negative latencies represent delay slots but list scheduler cannot utilize this information
- Usage of retargetable mixedBT backtracking scheduler produces up to 20% cycle count improvement compared to list scheduler
- Cycle count improvement of mixedBT scheduler is up to 7% better than existing listBT schedulers
- MixedBT is efficient because it behaves like a list scheduler for all instructions that do not have delay slots

Thank you !