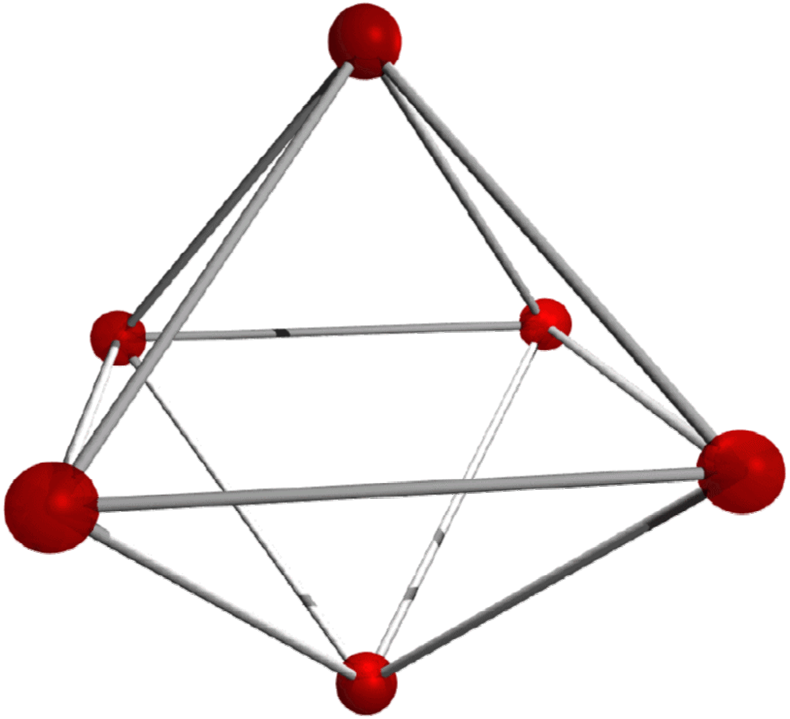
Worst Case Execution Time Prediction



AbsInt

Angewandte Informatik GmbH

Hard Real-Time Systems

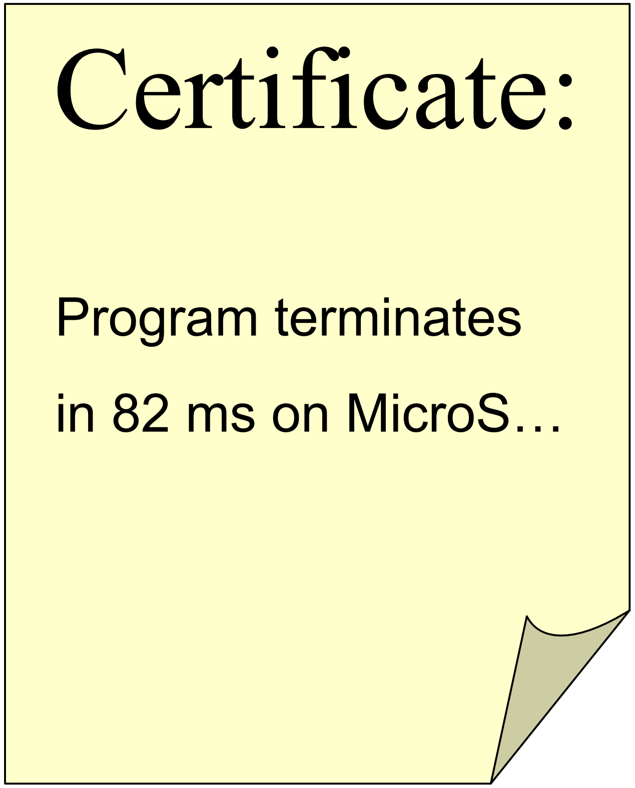
- Controllers in planes, cars, plants, … are expected to finish their tasks within reliable time bounds.

Timing Validation

- Schedulability analysis has to show that all timing requirements will be met
  - Takes into account:
    - System design (event based, time triggered, ...)
    - Outside world (maximal arrival rates, minimal interval between events, ...)
    - Scheduling policy (round robin, RMA, time triggered, RTOS, ...)
    - ...
- All results from the Scheduling Theory require the Worst Case Execution Time (WCET) of the tasks to be known

Certification

- Certificates
- Stand alone tool for e.g. TÜVs
- To proof timings to obtain certificates from TÜVs

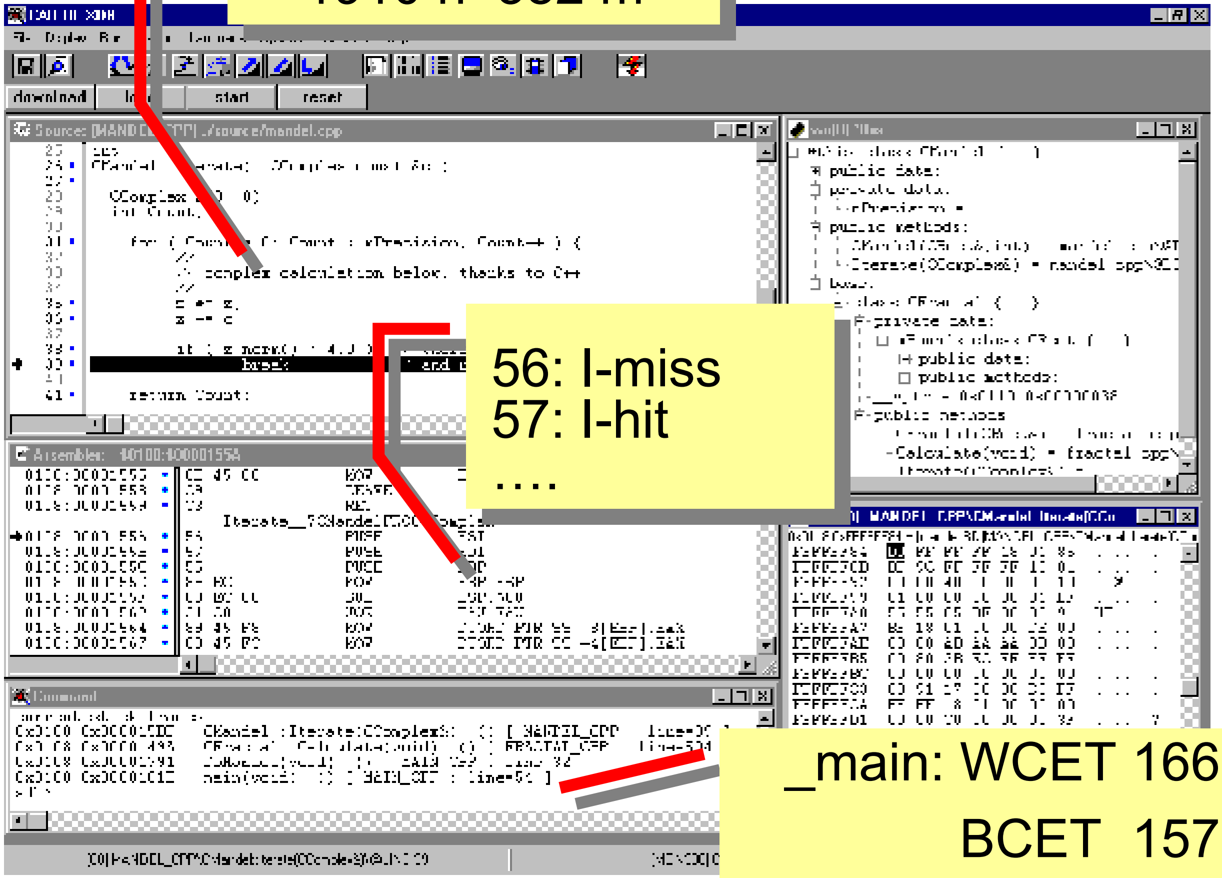


Certificate:

Program terminates

in 82 ms on MicroS…

Support during SW-Development



Source: [MANDEL_CPP] ./source/mandel.cpp

```
  CComplex z(0, 0);
  int Count;

    for ( Count = 0; Count < nPrecision; Count++ ) {
        //
        // complex calculation below, thanks to C++
        //
        z *= z;
        z += c;

        if ( z.norm() > 4.0 )
                   break;

    return Count;
```

Assembler: 0100:0000155A

Loop L31: IC=191
1910 h 382 m

56: I-miss
57: I-hit
....

_main: WCET 166.2 ms
BCET 157.0 ms

Modern Hardware

- Multiple memories, caches, pipelines, branch prediction, …
- Performance depends on execution history. This makes the prediction difficult
- No information means: assume the worst
- Switching off caching reduces performance by a factor of 30 (EADS study)

General Problems with State Based Processor Features

- Problem: Modern hardware <=> predictability of execution time

- Software monitoring, dual loop benchmark, direct measurement with logic analyzer, hardware simulation are no longer generally applicable.

- Choosing the fastest available processor, praying, or crossing fingers is not a true alternative.

A Traditional Approach

- Partition the application into code snippets,
- Determine their worst-case inputs,
- Measure their runtime with these inputs,
- Combine these results to find the worst-case path and its runtime.

- Error-prone and expensive!

Some Architectural Challenges

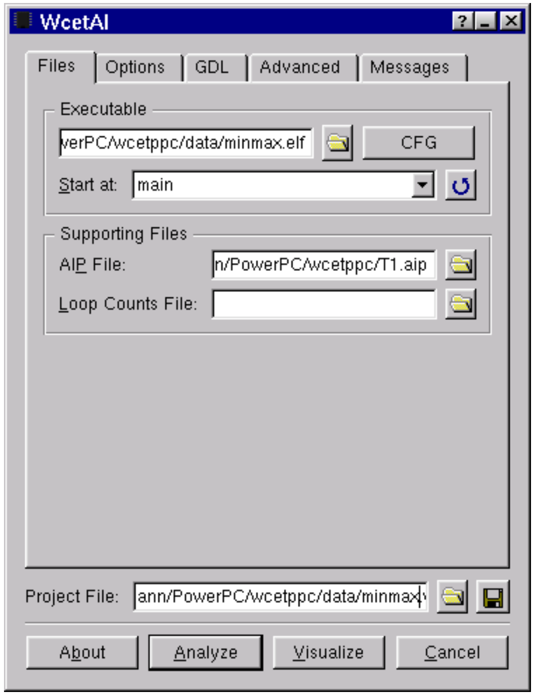
- The empty cache is not necessarily the "worst case cache"
- The global round robin counter/PLRU state bits can be changed by interrupt routines
- Unified cache => instructions and data interfere => measurements for all possibilities of interference necessary
- A cache miss is not necessarily the worst case

Solution: Static WCET Analysis

- The WCET analyzer computes save upper bounds of the execution time of the tasks in a program for all inputs
- Static program analysis based on Abstract Interpretation
- The analysis design is proven to be correct

WCET Analyzer

- Input: an executable program, starting points, loop iteration counts, call targets of indirect function calls, and a description of bus and memory speeds
- Computes **Worst-Case Execution Time** bounds of tasks

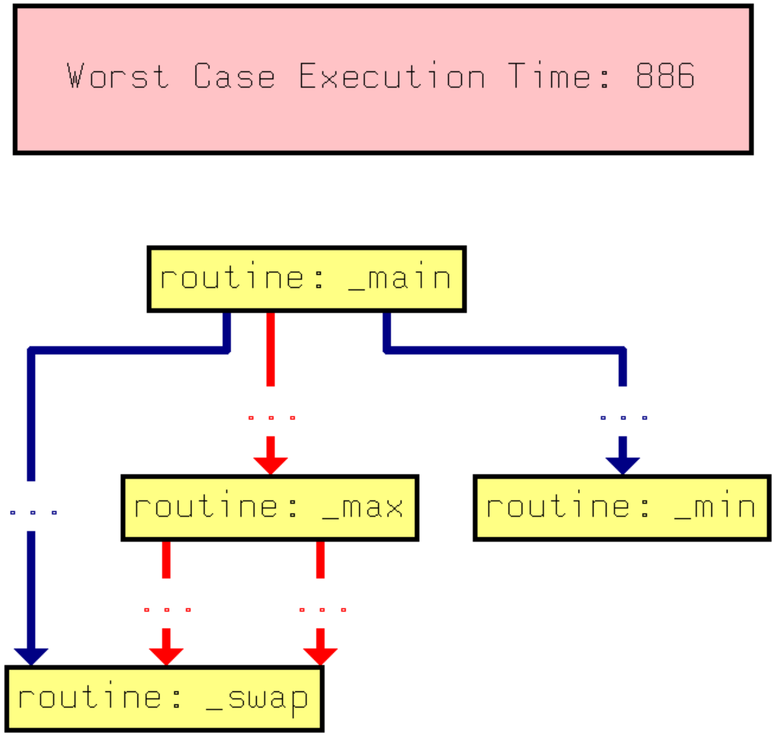


Executable
verPC/wcetppc/data/minmax.elf

Start at: main

Supporting Files
AIP File: n/PowerPC/wcetppc/T1.aip
Loop Counts File:

Project File: ann/PowerPC/wcetppc/data/minmax



Worst Case Execution Time: 886

Scope

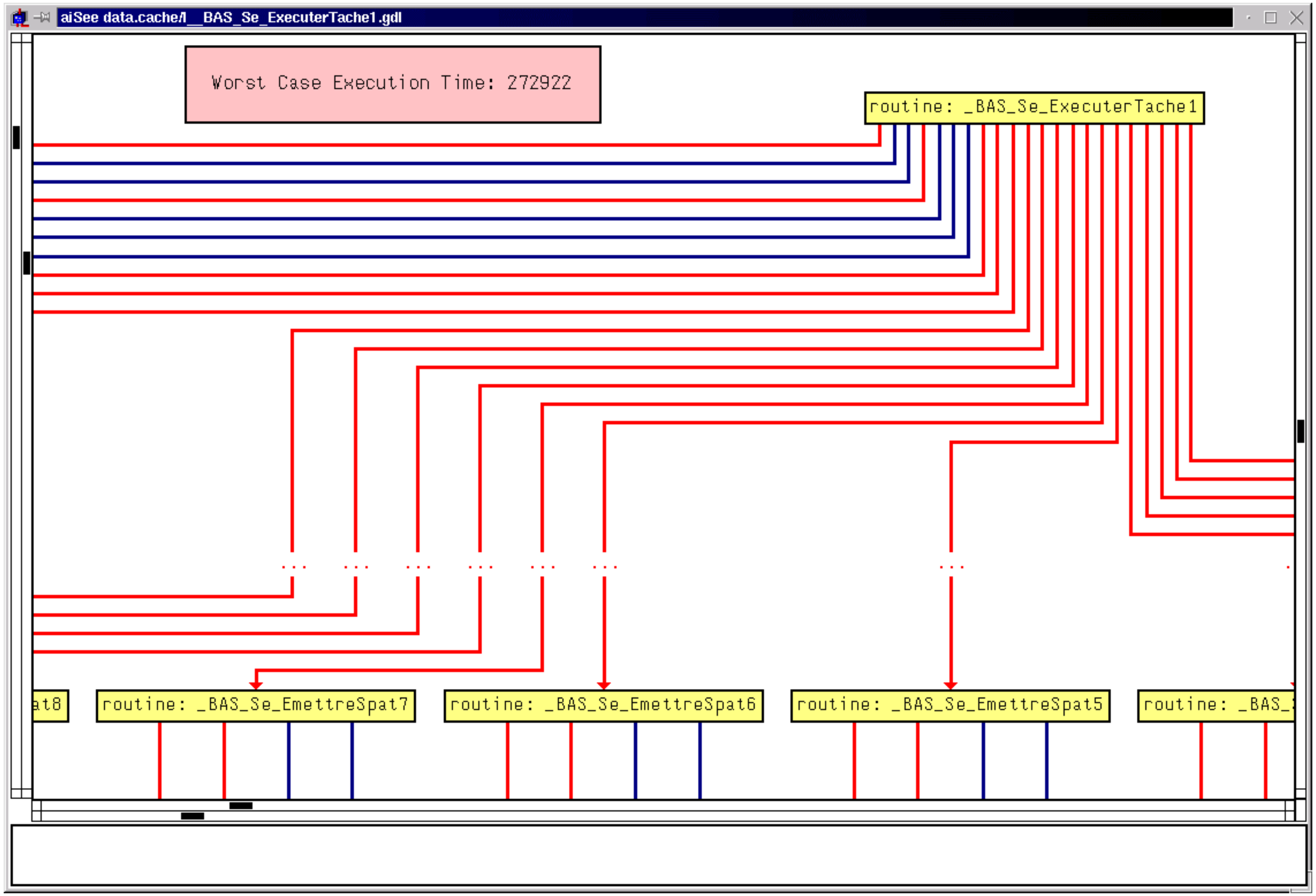
- The WCET analyzer assumes no interference from the outside. Effects of interrupts, IO, timers, other (co-) processors are not reflected in the predicted runtime and have to be considered separately.

Input

- An executable (e.g. in ELF or COFF format).
- User annotations
  - Call targets for all indirect function calls
  - Upper bounds on the iteration counts of all loops (and recursions)
- A description of the (external) memories and buses (i.e. a list of memory areas with minimal and maximal access times)
  - To be provided once for a new board
- A task
  - Can be arbitrarily selected by a start address or a function name
  - A task denotes a sequentially executed piece of code (no threads, no parallelism, no waiting for external events)
  - The code of a task is compiled by a C-compiler from a restricted subset of ANSI-C (no dynamic data structures, no setjmp/longjmp)

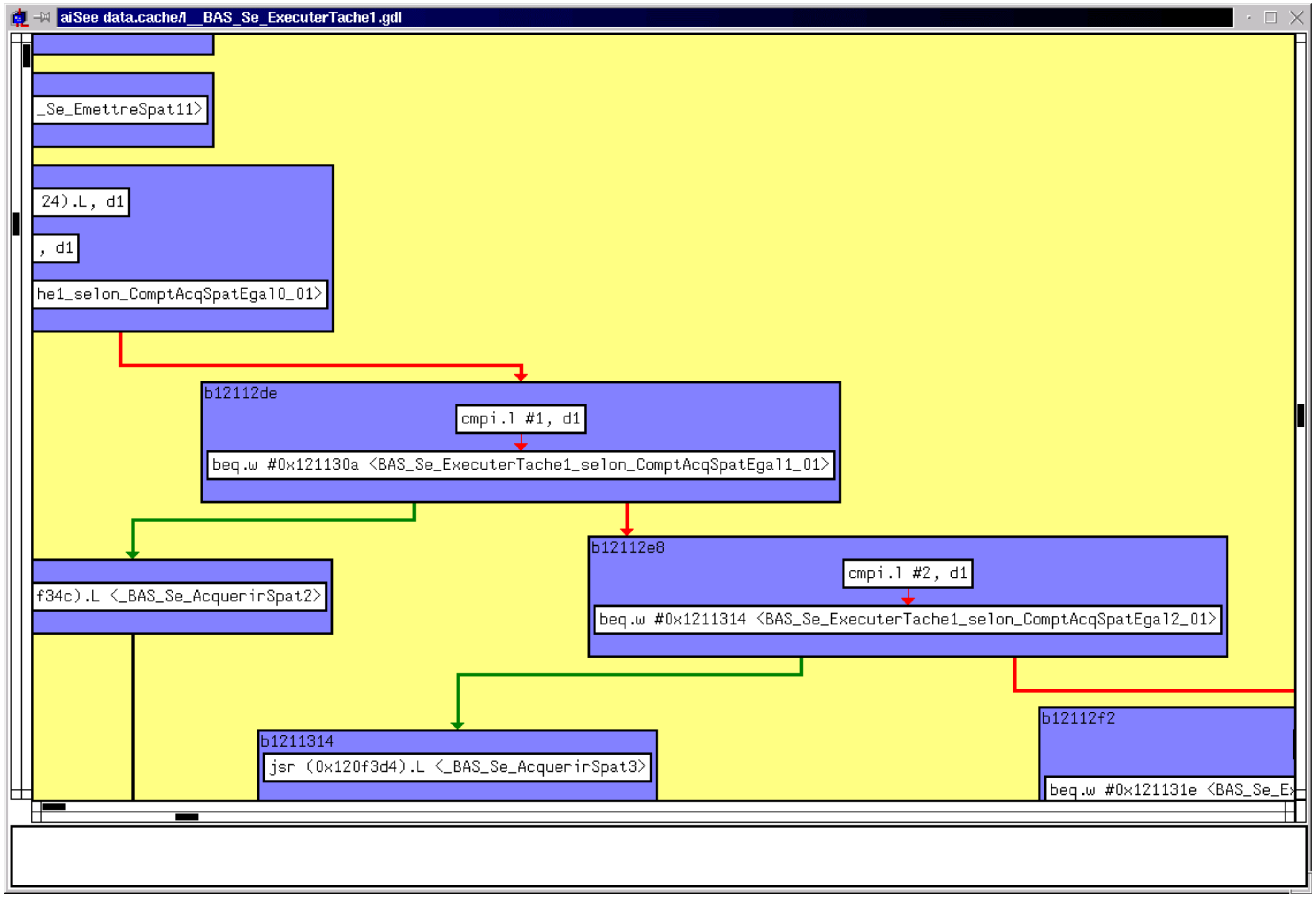
Call Graph

Calls contributing to the WCET are red


aiSee data.cache/l__BAS_Se_ExecuterTache1.gdl
Worst Case Execution Time: 272922
routine: _BAS_Se_ExecuterTache1
at8
routine: _BAS_Se_EmettreSpat7
routine: _BAS_Se_EmettreSpat6
routine: _BAS_Se_EmettreSpat5
routine: _BAS_

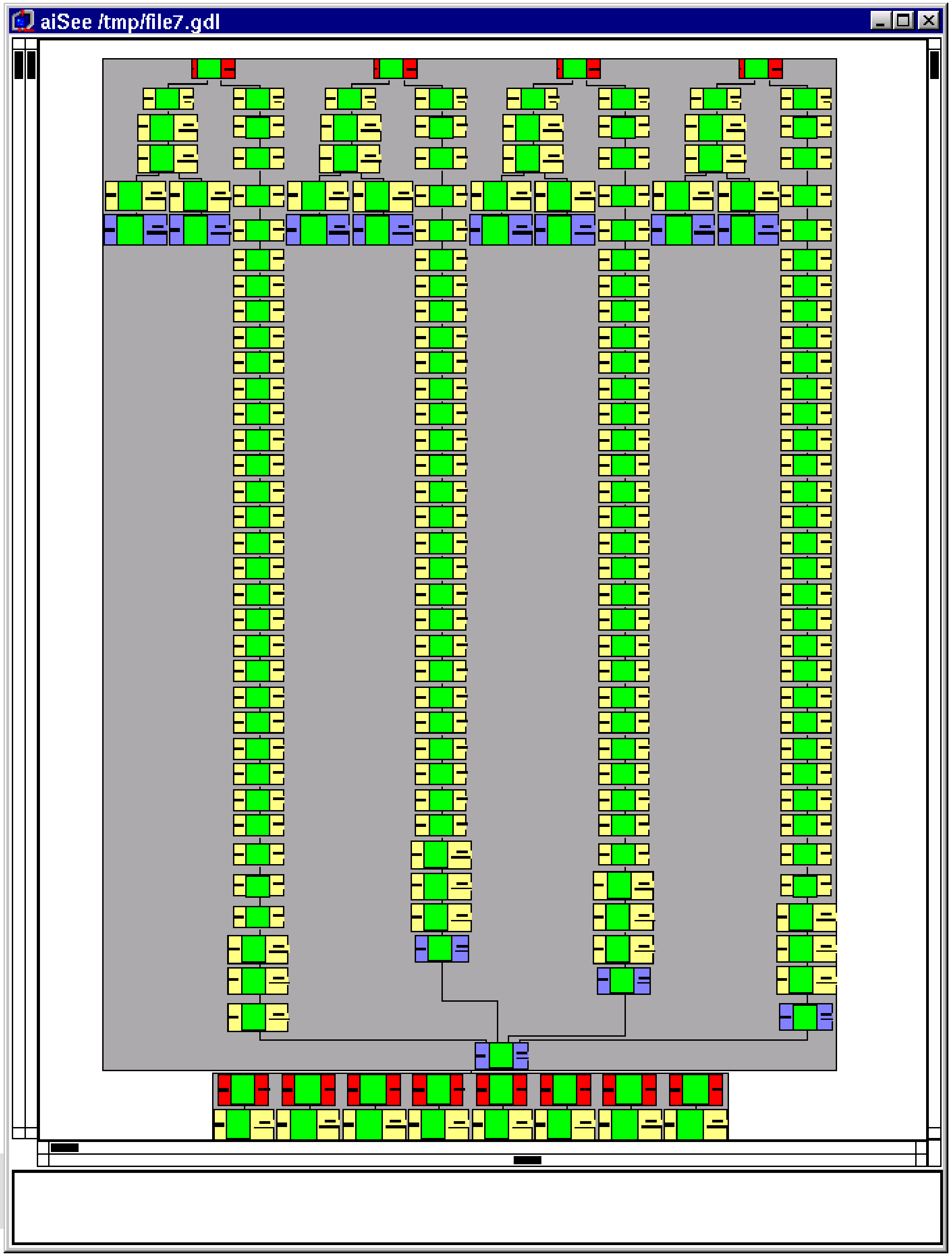
Control Flow Graph

Worst case path is red



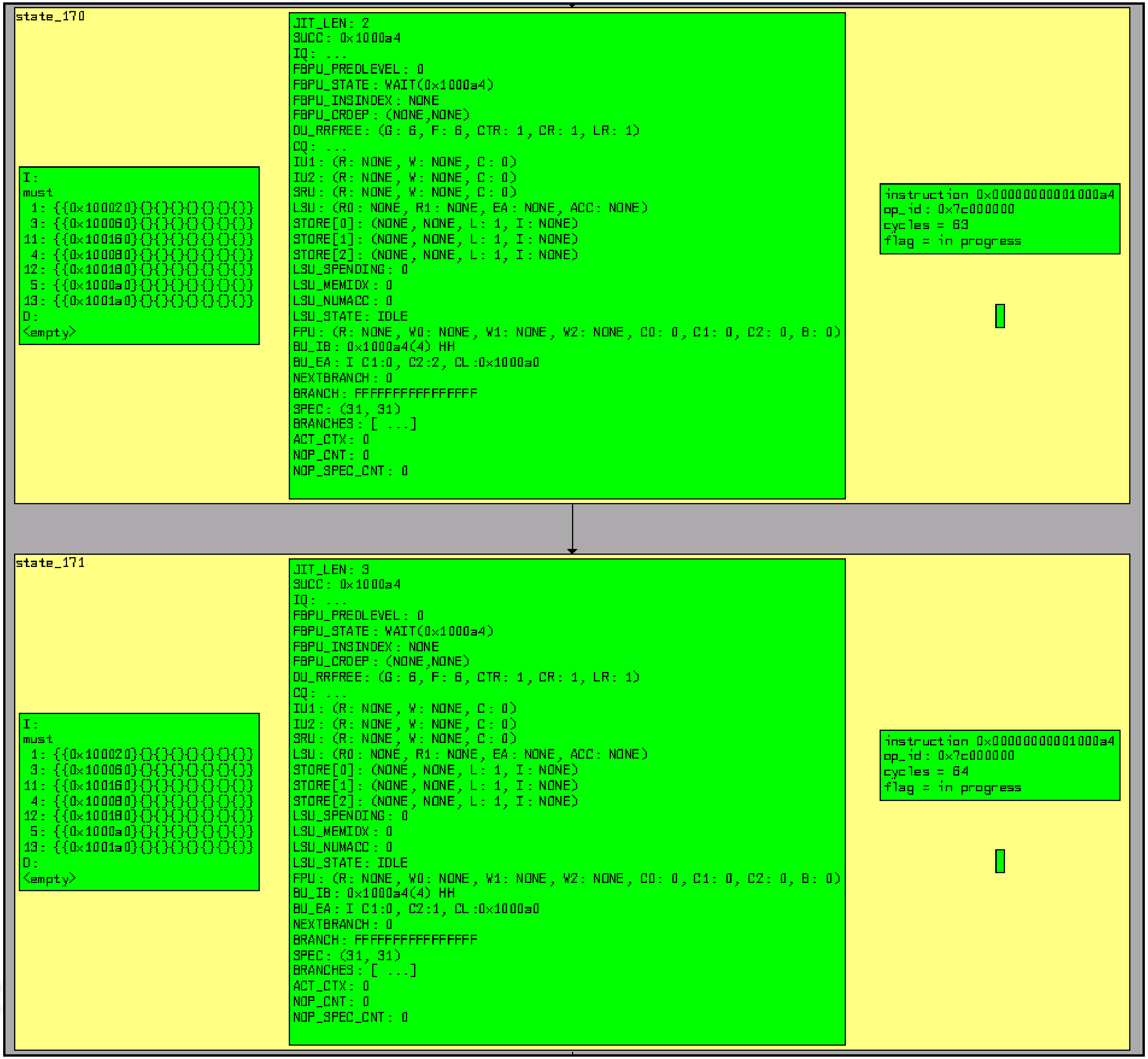
```
_Se_EmettreSpat11>

24).L, d1
, d1
he1_selon_ComptAcqSpatEgal0_01>

b12112de
cmpi.l #1, d1
beq.w #0x121130a <BAS_Se_ExecuterTache1_selon_ComptAcqSpatEgal1_01>

f34c).L <_BAS_Se_AcquerirSpat2>

b12112e8
cmpi.l #2, d1
beq.w #0x1211314 <BAS_Se_ExecuterTache1_selon_ComptAcqSpatEgal2_01>

b1211314
jsr (0x120f3d4).L <_BAS_Se_AcquerirSpat3>

b12112f2
beq.w #0x121131e <BAS_Se_E
```

Cycle-Wise Evolution of Cache/ Pipeline States



AbsInt
Angewandte Informatik

Cache/ Pipeline State



```
state_170

I:
must
 1: {{0x100020}{}{}{}{}{}{}{}}
 3: {{0x100060}{}{}{}{}{}{}{}}
11: {{0x100160}{}{}{}{}{}{}{}}
 4: {{0x100080}{}{}{}{}{}{}{}}
12: {{0x100180}{}{}{}{}{}{}{}}
 5: {{0x1000a0}{}{}{}{}{}{}{}}
13: {{0x1001a0}{}{}{}{}{}{}{}}
D:
<empty>
```

```
JIT_LEN: 2
SUCC: 0x1000a4
IQ: ...
FBPU_PREDLEVEL: 0
FBPU_STATE: WAIT(0x1000a4)
FBPU_INSINDEX: NONE
FBPU_CRDEP: (NONE,NONE)
DU_RRFREE: (G: 6, F: 6, CTR: 1, CR: 1, LR: 1)
CQ: ...
IU1: (R: NONE, W: NONE, C: 0)
IU2: (R: NONE, W: NONE, C: 0)
SRU: (R: NONE, W: NONE, C: 0)
LSU: (R0: NONE, R1: NONE, EA: NONE, ACC: NONE)
STORE[0]: (NONE, NONE, L: 1, I: NONE)
STORE[1]: (NONE, NONE, L: 1, I: NONE)
STORE[2]: (NONE, NONE, L: 1, I: NONE)
LSU_SPENDING: 0
LSU_MEMIDX: 0
LSU_NUMACC: 0
LSU_STATE: IDLE
FPU: (R: NONE, W0: NONE, W1: NONE, W2: NONE, C0: 0, C1: 0, C2: 0, B: 0)
BU_IB: 0x1000a4(4) HH
BU_EA: I C1:0, C2:2, CL:0x1000a0
NEXTBRANCH: 0
BRANCH: FFFFFFFFFFFFFFFF
SPEC: (31, 31)
BRANCHES: [ ...]
ACT_CTX: 0
NOP_CNT: 0
NOP_SPEC_CNT: 0
```

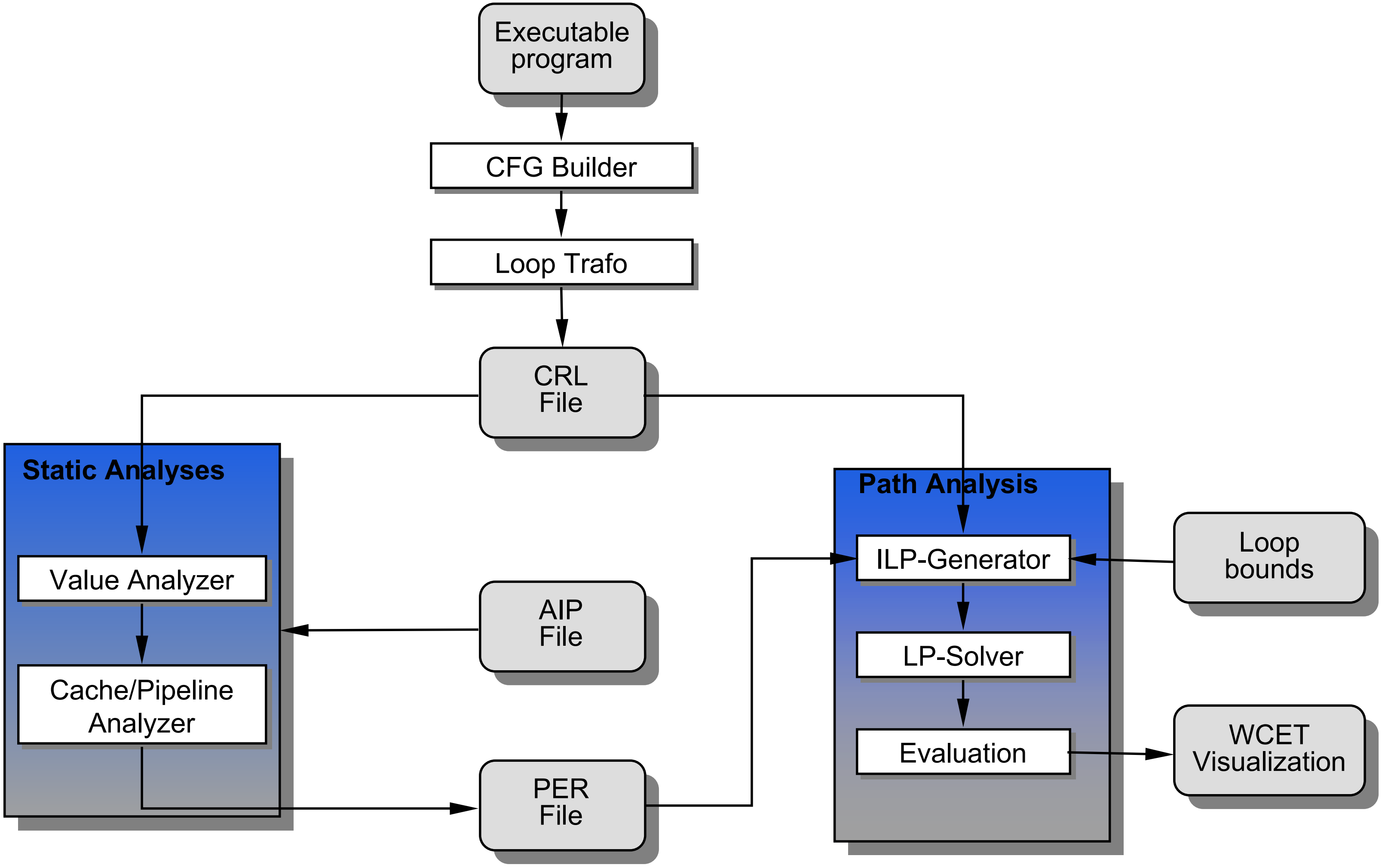
```
instruction 0x00000000001000a4
op_id: 0x7c000000
cycles = 63
flag = in progress
```

```
state_171

I:
must
 1: {{0x100020}{}{}{}{}{}{}{}}
 3: {{0x100060}{}{}{}{}{}{}{}}
11: {{0x100160}{}{}{}{}{}{}{}}
 4: {{0x100080}{}{}{}{}{}{}{}}
12: {{0x100180}{}{}{}{}{}{}{}}
 5: {{0x1000a0}{}{}{}{}{}{}{}}
13: {{0x1001a0}{}{}{}{}{}{}{}}
D:
<empty>
```

```
JIT_LEN: 3
SUCC: 0x1000a4
IQ: ...
FBPU_PREDLEVEL: 0
FBPU_STATE: WAIT(0x1000a4)
FBPU_INSINDEX: NONE
FBPU_CRDEP: (NONE,NONE)
DU_RRFREE: (G: 6, F: 6, CTR: 1, CR: 1, LR: 1)
CQ: ...
IU1: (R: NONE, W: NONE, C: 0)
IU2: (R: NONE, W: NONE, C: 0)
SRU: (R: NONE, W: NONE, C: 0)
LSU: (R0: NONE, R1: NONE, EA: NONE, ACC: NONE)
STORE[0]: (NONE, NONE, L: 1, I: NONE)
STORE[1]: (NONE, NONE, L: 1, I: NONE)
STORE[2]: (NONE, NONE, L: 1, I: NONE)
LSU_SPENDING: 0
LSU_MEMIDX: 0
LSU_NUMACC: 0
LSU_STATE: IDLE
FPU: (R: NONE, W0: NONE, W1: NONE, W2: NONE, C0: 0, C1: 0, C2: 0, B: 0)
BU_IB: 0x1000a4(4) HH
BU_EA: I C1:0, C2:1, CL:0x1000a0
NEXTBRANCH: 0
BRANCH: FFFFFFFFFFFFFFFF
SPEC: (31, 31)
BRANCHES: [ ...]
ACT_CTX: 0
NOP_CNT: 0
NOP_SPEC_CNT: 0
```

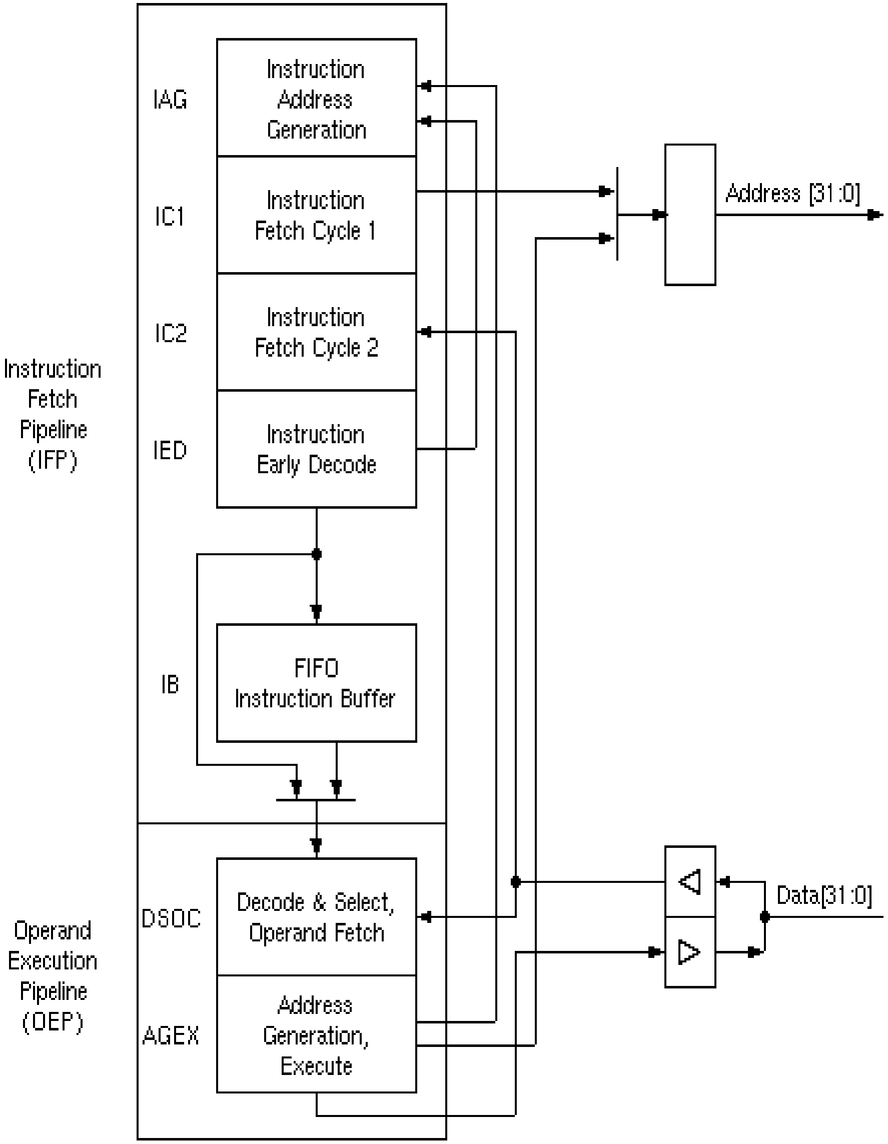
```
instruction 0x00000000001000a4
op_id: 0x7c000000
cycles = 64
flag = in progress
```

Overall Structure

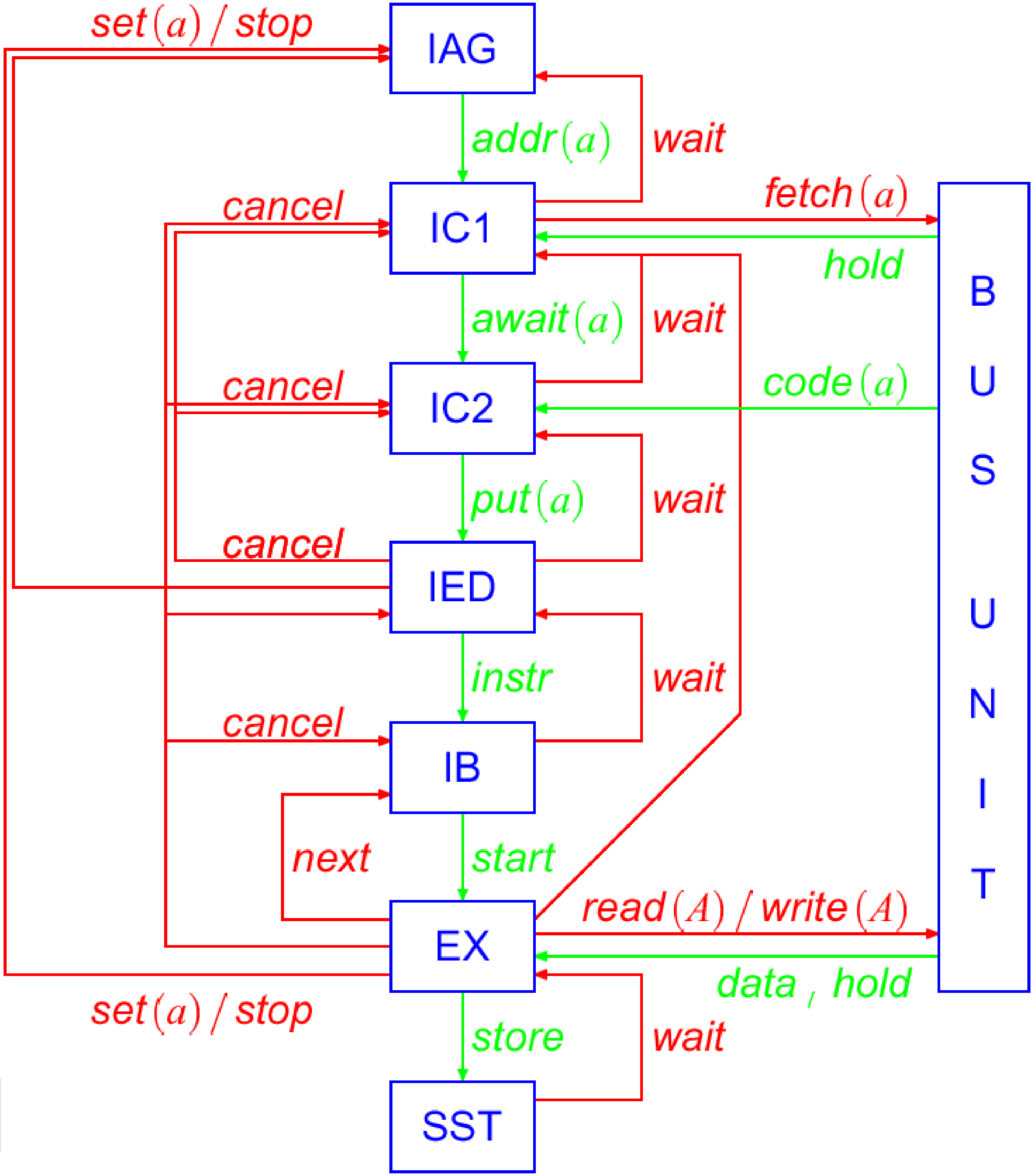


The ColdFire Pipeline

- Fetch Pipeline of 4 stages
  - Instruction Address Generation (IAG)
  - Instruction Fetch Cycle 1 (IC1)
  - Instruction Fetch Cycle 2 (IC2)
  - Instruction Early Decode (IED)
- Instruction Buffer (IB) for 8 instructions
- Execution Pipeline of 2 stages
  - Decoding and register operand fetching (1 cycle)
  - Memory access and execution (1 – many cycles)


IAG
Instruction Address Generation
IC1
Instruction Fetch Cycle 1
IC2
Instruction Fetch Cycle 2
IED
Instruction Early Decode
Instruction Fetch Pipeline (IFP)
IB
FIFO Instruction Buffer
DSOC
Decode & Select, Operand Fetch
AGEX
Address Generation, Execute
Operand Execution Pipeline (OEP)
Address [31:0]
Data[31:0]

Pipeline Model

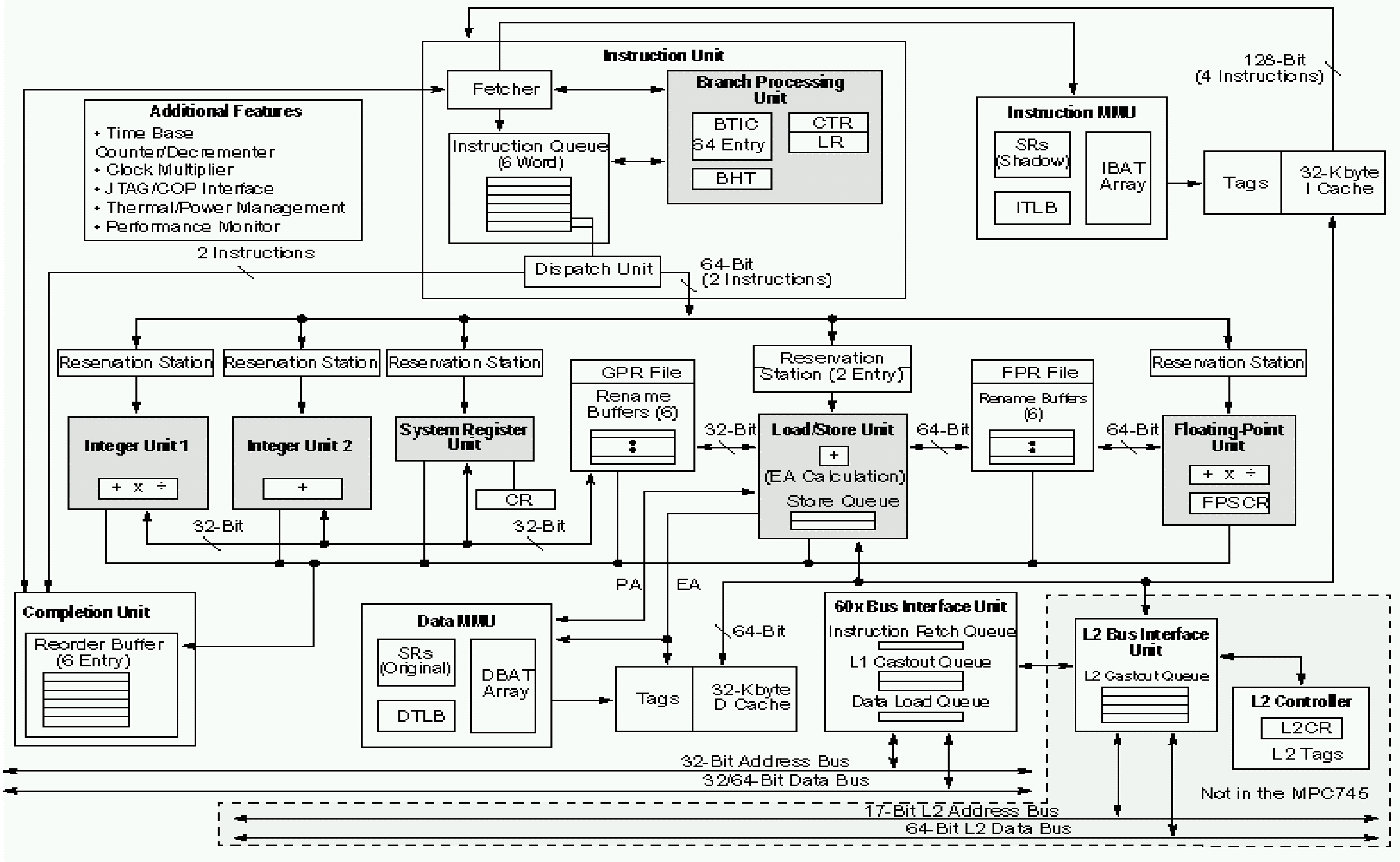


AbsInt
Angewandte Informatik

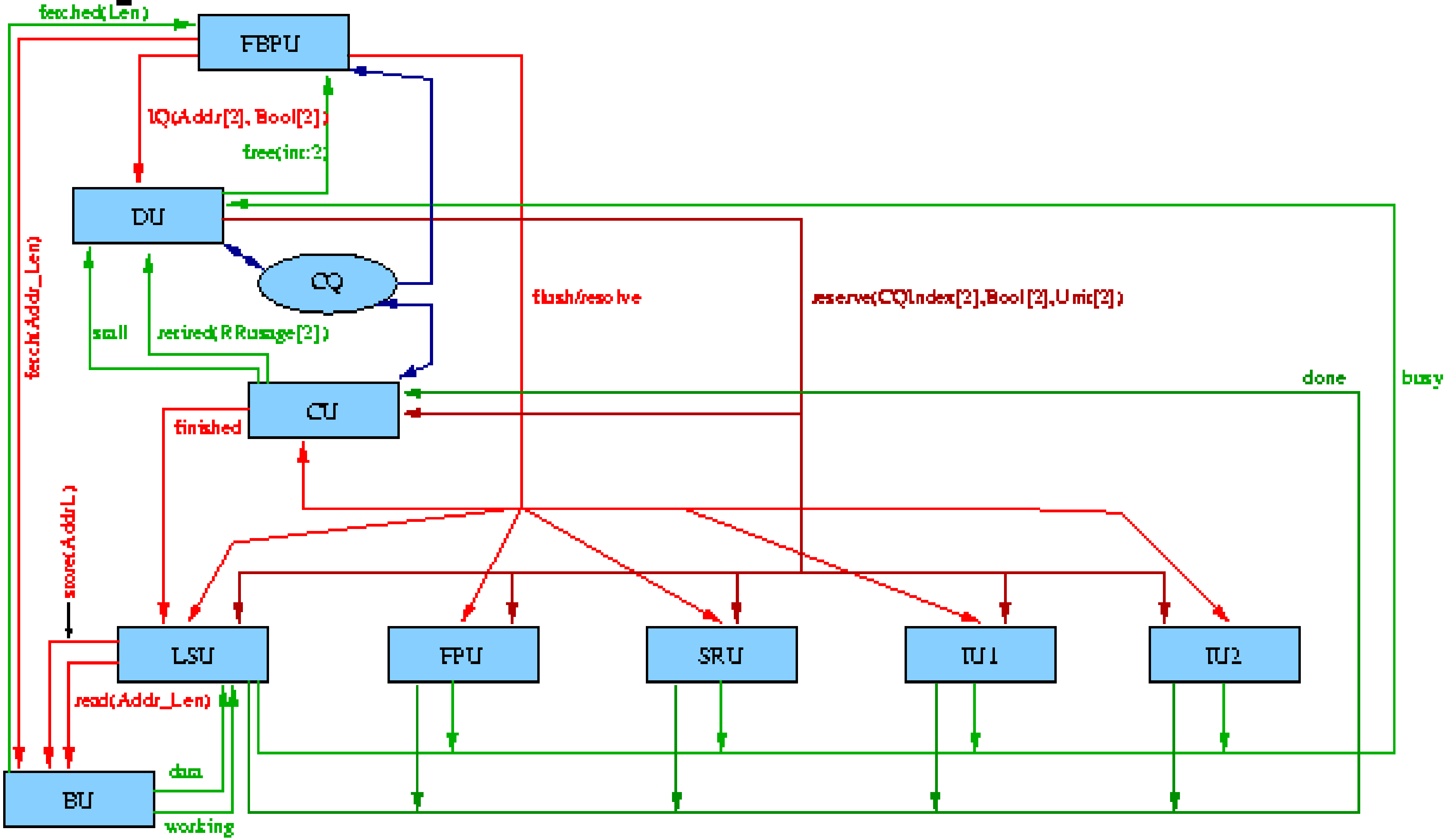
PPC755 Pipeline

- Complex branch prediction
- Superscalar: up to two instructions per cycle dispatched and retired
- Out-of-order execution
  - Integer instructions,
  - Floating point instructions and
  - Loads may be reordered
- Speculative execution
  - After predicted branches, instructions are executed speculatively
  - Speculative loads may occur

Pipeline of the PPC755

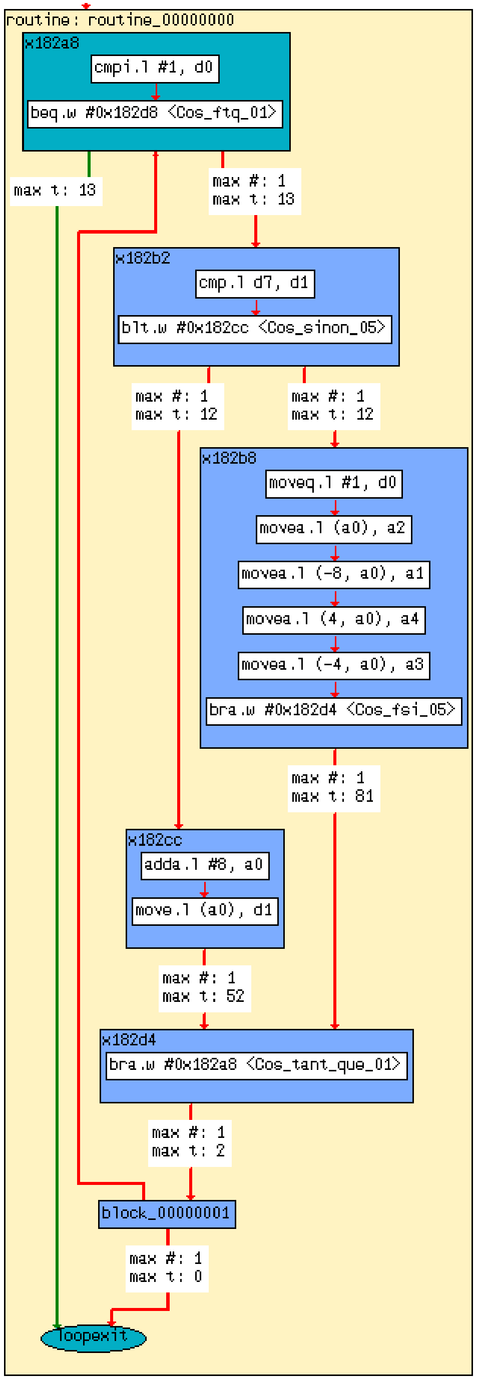


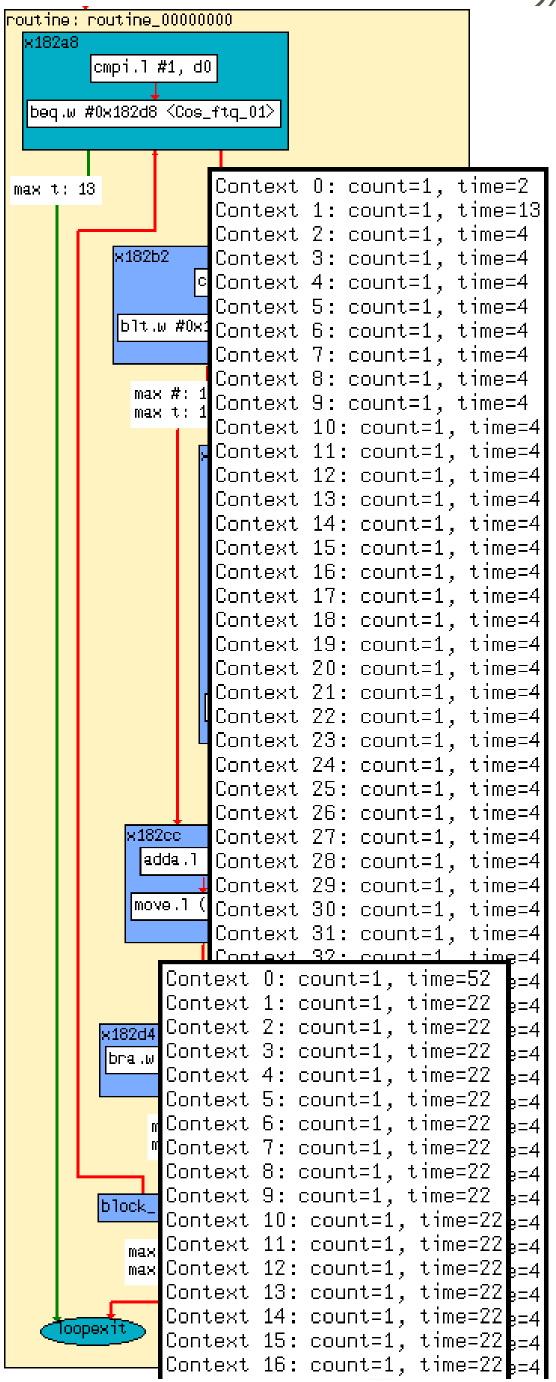
Pipeline Model



Analysis of Loops

- Loops are analyzed like procedures
- This allows for
  - Virtual inlining
  - Virtual unrolling
  - Better address resolution
  - Burst accesses
  - Selectable precision
- Optional user constraints





```
Context  0: count=1, time=2
Context  1: count=1, time=13
Context  2: count=1, time=4
Context  3: count=1, time=4
Context  4: count=1, time=4
Context  5: count=1, time=4
Context  6: count=1, time=4
Context  7: count=1, time=4
Context  8: count=1, time=4
Context  9: count=1, time=4
Context  10: count=1, time=4
Context  11: count=1, time=4
Context  12: count=1, time=4
Context  13: count=1, time=4
Context  14: count=1, time=4
Context  15: count=1, time=4
Context  16: count=1, time=4
Context  17: count=1, time=4
Context  18: count=1, time=4
Context  19: count=1, time=4
Context  20: count=1, time=4
Context  21: count=1, time=4
Context  22: count=1, time=4
Context  23: count=1, time=4
Context  24: count=1, time=4
Context  25: count=1, time=4
Context  26: count=1, time=4
Context  27: count=1, time=4
Context  28: count=1, time=4
Context  29: count=1, time=4
Context  30: count=1, time=4
Context  31: count=1, time=4
Context  32: count=1, time=4
```

```
Context  0: count=1, time=52
Context  1: count=1, time=22
Context  2: count=1, time=22
Context  3: count=1, time=22
Context  4: count=1, time=22
Context  5: count=1, time=22
Context  6: count=1, time=22
Context  7: count=1, time=22
Context  8: count=1, time=22
Context  9: count=1, time=22
Context  10: count=1, time=22
Context  11: count=1, time=22
Context  12: count=1, time=22
Context  13: count=1, time=22
Context  14: count=1, time=22
Context  15: count=1, time=22
Context  16: count=1, time=22
```

Limitations

- Well behaved code (ABI)
- No exceptions
- Only aligned accesses
- No VM settings
- Some instructions not to be used

Advantages

- The WCET analyzer allows you to:
    - inspect the timing behavior of (timing critical parts of) your code
- The analysis results
    - are determined without the need to change the code
    - hold for all executions (for the intrinsic cache and pipeline behavior)

Advantages

- The results are **precise**
- The computation is **fast**
- The WCET analyzer is **easy to use**
- The WCET analyzer works on **optimized code**
- The WCET analyzer **saves development time** by avoiding the tedious and time consuming (instrument and) execute (or emulate) and measure cycle for a set of inputs over and over again

Planned Extensions

- Support for code generators
  - E.g. recognition of generated patterns to avoid the need of user annotations on generated code
- Automatic detection of the number of loop iterations in the executable
- Further targets

Support for a New Processor Model

- Front end for executables
- Modeling of the pipeline according to the documentation
- (Modeling of the cache)
- Verification of the pipeline model on the real hardware or reliable emulator

References

- LCTRTS’ 97. Ferdinand, Martin, and Wilhelm. Applying Compiler Techniques to Cache Behavior Prediction.
- RTSS’ 98. Theiling and Ferdinand. Combining Abstract Interpretation and ILP for Microarchitecture Modeling and Program Path Analysis.
- STTT Journal. Martin. PAG -- An Efficient Program Analyzer Generator
- LCTES’ 99. Schneider and Ferdinand. Pipeline Behavior Prediction for Superscalar Processors by Abstract Interpretation.
- RTS Journal. Ferdinand and Wilhelm. Efficient and Precise Cache Behavior Prediction for Real-Time Systems.
- RTS Journal. Kästner and Thesing. Cache-Aware Pre-Runtime Scheduling.
- RTS Journal. Theiling and Ferdinand. Fast and Precise WCET Prediction by Separated Cache and Path Analysis.
- RTSS ´00. Schneider. Cache and Pipeline Sensitive Fixed Priority Scheduling for Preemptive Real-Time Systems.
- RTCSA´ 00. Theiling. Extracting Safe and Precise Control Flow from Binaries
- LCTES’ 01. Theiling. Generating Decision Trees for Decoding Binaries
- EMSOFT’ 01. Ferdinand et al., Reliable and Precise WCET Determination for a Real-Life Procesor
- **MSP’ 02. Heckmann. The Influence of Replacement Strategy on Static Prediction of Cache Contents**
- **ECRTS’ 02. Thesing, Langenbach, Heckmann. Pipeline Behavior Prediction Analysis for Real-Time Systems by Pipeline Modeling**
- **EMSOFT’ 02. Theiling. ILP-based Interprocedural Path Analysis**
- **SAS’ 02. Langenbach, Thesing, Heckmann. Pipeline Modeling for Timing Analysis**
- **WCET’ 02. Langenbach, Ferdinand, Wilhelm. Worst Case Execution Time Prediction**

AbsInt

Angewandte Informatik

email: info@AbsInt.com

http://www.AbsInt.com