Timed Games for Computing WCET on Pipelined Processors with Caches

Franck Cassez

CNRS & Marie Curie Fellow
IRCCyN, Nantes, France

June 24th, 2011
ACSD'2011, Newcastle, UK

Outline of the talk

1. The Worst-Case Execution-Time Problem
2. Modular Computation of WCET
3. Experiments & Results
4. Conclusion & Future Work

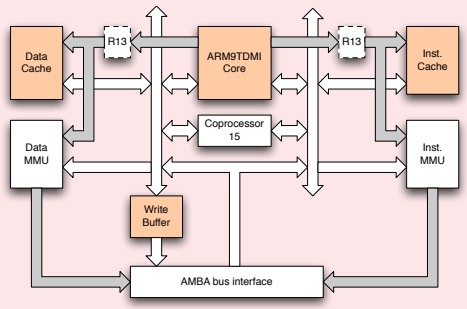
Worst-Case Execution-Time

Input data
$d \in \mathcal{D}$

Program P

```
10: e3a03000    mov   r3, #0
14: e58d3014    str   r3, [sp, #20]
18: e3a03002    mov   r3, #2
1c: e58d300c    str   r3, [sp, #12]
20: ea00000a    b     50 <fib+0x50>
24: e59d3010    ldr   r3, [sp, #16]
28: e58d3018    str   r3, [sp, #24]
2c: e59d2010    ldr   r2, [sp, #16]
30: e59d3014    ldr   r3, [sp, #20]
34: e0823003    add   r3, r2, r3
38: e58d3010    str   r3, [sp, #16]
3c: e59d3018    ldr   r3, [sp, #24]
40: e58d3014    str   r3, [sp, #20]
44: e59d300c    ldr   r3, [sp, #12]
48: e2833001    add   r3, r3, #1
4c: e58d300c    str   r3, [sp, #12]
50: e59d200c    ldr   r2, [sp, #12]
54: e59d3004    ldr   r3, [sp, #4]
58: e1520003    cmp   r2, r3
5c: dafffff0    ble   24 <fib+0x24>
```

Hardware H



→ time(H, P, d)

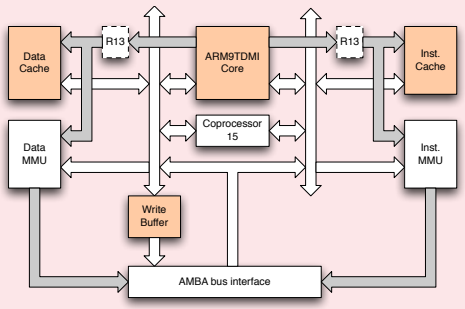
Worst-Case Execution-Time

Input data
$d \in \mathcal{D}$

Program P

```
10: e3a03000    mov   r3, #0
14: e58d3014    str   r3, [sp, #20]
18: e3a03002    mov   r3, #2
1c: e58d300c    str   r3, [sp, #12]
20: ea00000a    b     50 <fib+0x50>
24: e59d3010    ldr   r3, [sp, #16]
28: e58d3018    str   r3, [sp, #24]
2c: e59d2010    ldr   r2, [sp, #16]
30: e59d3014    ldr   r3, [sp, #20]
34: e0823003    add   r3, r2, r3
38: e58d3010    str   r3, [sp, #16]
3c: e59d3018    ldr   r3, [sp, #24]
40: e58d3014    str   r3, [sp, #20]
44: e59d300c    ldr   r3, [sp, #12]
48: e2833001    add   r3, r3, #1
4c: e58d300c    str   r3, [sp, #12]
50: e59d200c    ldr   r2, [sp, #12]
54: e59d3004    ldr   r3, [sp, #4]
58: e1520003    cmp   r2, r3
5c: dafffff0    ble   24 <fib+0x24>
```

Hardware H



→ time(H, P, d)

$$WCET(H, P) = \max_{d \in \mathcal{D}} time(H, P, d)$$

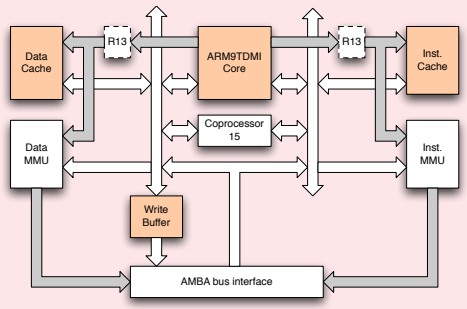
Worst-Case Execution-Time

Input data
$d \in \mathcal{D}$

Program P

```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  dafffff0    ble   24 <fib+0x24>
```

Hardware H



→ time(H, P, d)

$$WCET(H,P) \leq WCET\text{-}UB(H,P) \leq (1+\varepsilon) \times WCET(H,P)$$

Related Work & Existing Methods

Partial – Tests/Simulation

- random
- probabilistic
- real board, simulator

- easy to implement
- not exhaustive
- not safe: gives a lower bound

Tools: RapiTime (based on pWCET) and Mtime

Exhaustive – Static Analysis & Integer Linear Programming

1. Compute a control flow graph of P
2. Determine loop upper bounds
3. Build a weighted CFG
4. Solve an integer linear program

- harder to implement
- safe: gives an upper bound
- manual annotations
- algorithm is monolithic

Tools: Bound-T, OTAWA, TuBound, Chronos, SWEET and aiT (AbsInt)

Related Work & Existing Methods

Partial - Tests/Simulation

- random
- probabilistic
- real board, simulator

- easy to implement
- not exhaustive
- not safe: gives a lower bound

Tools: RapiTime (based on pWCET) and Mtime

Exhaustive - Static Analysis & Integer Linear Programming

1. Compute a control flow graph of P
2. Determine loop upper bounds
3. Build a weighted CFG
4. Solve an integer linear program

- harder to implement
- safe: gives an upper bound
- manual annotations
- algorithm is monolithic

Tools: Bound-T, OTAWA, TuBound, Chronos, SWEET and aiT (AbsInt)

Our Contribution

Assumptions on binary program P:

- termination of P does not depend on input data
- P always terminates

Results

- Fully automatic computation of WCET
  computation of CFG (and stack size)
  computation of WCET-equivalent program
- Modular method
  1. Program model
  2. Hardware model
  3. Analysis (computation of WCET)
- Comparison of computed WCET and actual WCET
  WCET Benchmarks from Mälardalen University
  measurements on real platform ARM920T
  WCET computed with UPPAAL

[Bec11] J.-L. Béchennec and F. Cassez,
Computation of WCET using Program Slicing and Real-Time Model-Checking,
Research report, IRCCyN/CNRS, May 2011, arXiv:1105.1633v2 [cs.SE].

Our Contribution

Assumptions on binary program P:

- termination of P does not depend on input data
- P always terminates

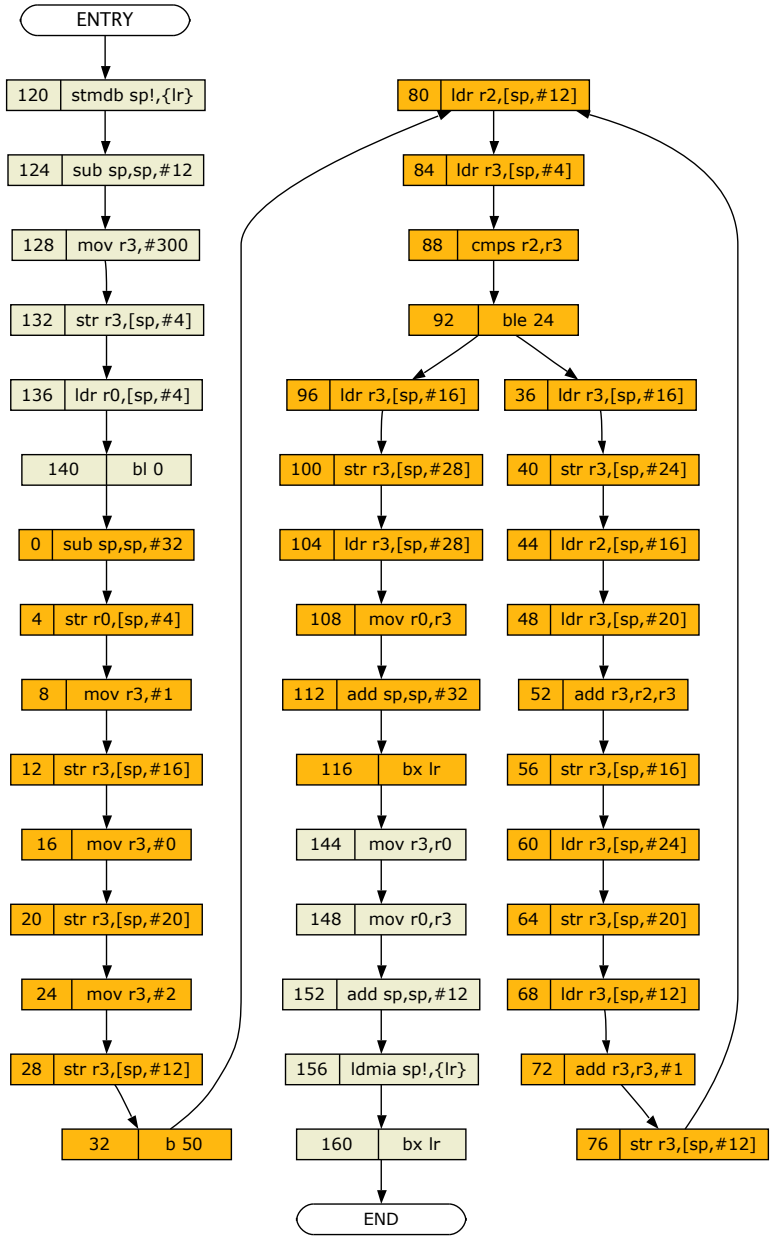
Results

- Fully automatic computation of WCET
  computation of CFG (and stack size)
  computation of WCET-equivalent program
- Modular method
  1. Program model
  2. Hardware model
  3. Analysis (computation of WCET)
- Comparison of computed WCET and actual WCET
  WCET Benchmarks from Mälardalen University
  measurements on real platform ARM920T
  WCET computed with UPPAAL

[Bec11] J.-L. Béchennec and F. Cassez,
Computation of WCET using Program Slicing and Real-Time Model-Checking,
Research report, IRCCyN/CNRS, May 2011, arXiv:1105.1633v2 [cs.SE].

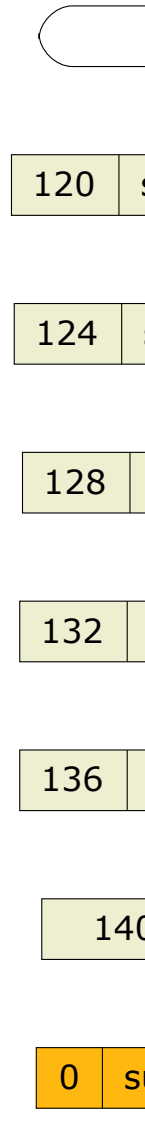
The Fibonacci Program

```
00000000 <fib>:
   0:  e24dd020     sub   sp, sp, #32
   4:  e58d0004     str   r0, [sp, #4]
   8:  e3a03001     mov   r3, #1
   c:  e58d3010     str   r3, [sp, #16]
  10:  e3a03000     mov   r3, #0
  14:  e58d3014     str   r3, [sp, #20]
  18:  e3a03002     mov   r3, #2
  1c:  e58d300c     str   r3, [sp, #12]
  20:  ea00000a     b     50 <fib+0x50>
  24:  e59d3010     ldr   r3, [sp, #16]
  28:  e58d3018     str   r3, [sp, #24]
  2c:  e59d2010     ldr   r2, [sp, #16]
  30:  e59d3014     ldr   r3, [sp, #20]
  34:  e0823003     add   r3, r2, r3
  38:  e58d3010     str   r3, [sp, #16]
  3c:  e59d3018     ldr   r3, [sp, #24]
  40:  e58d3014     str   r3, [sp, #20]
  44:  e59d300c     ldr   r3, [sp, #12]
  48:  e2833001     add   r3, r3, #1
  4c:  e58d300c     str   r3, [sp, #12]
  50:  e59d200c     ldr   r2, [sp, #12]
  54:  e59d3004     ldr   r3, [sp, #4]
  58:  e1520003     cmp   r2, r3
  5c:  dafffff0     ble   24 <fib+0x24>
  60:  e59d3010     ldr   r3, [sp, #16]
  64:  e58d301c     str   r3, [sp, #28]
  68:  e59d301c     ldr   r3, [sp, #28]
  6c:  e1a00003     mov   r0, r3
  70:  e28dd020     add   sp, sp, #32
  74:  e12fff1e     bx    lr

00000078 <main>:
  78:  e52de004     push  {lr}
  7c:  e24dd00c     sub   sp, sp, #12
  80:  e3a03f4b     mov   r3, #300
  84:  e58d3004     str   r3, [sp, #4]
  88:  e59d0004     ldr   r0, [sp, #4]
  8c:  ebffffdb     bl    0 <fib>
  90:  e1a03000     mov   r3, r0
  94:  e1a00003     mov   r0, r3
  98:  e28dd00c     add   sp, sp, #12
  9c:  e49de004     pop   {lr}
  a0:  e12fff1e     bx    lr
```



The Fibonacci Program

```
00000000 <fib>:
   0:  e24dd020     sub   sp, sp, #32
   4:  e58d0004     str   r0, [sp, #4]
   8:  e3a03001     mov   r3, #1
   c:  e58d3010     str   r3, [sp, #16]
  10:  e3a03000     mov   r3, #0
  14:  e58d3014     str   r3, [sp, #20]
  18:  e3a03002     mov   r3, #2
  1c:  e58d300c     str   r3, [sp, #12]
  20:  ea00000a     b     50 <fib+0x50>
  24:  e59d3010     ldr   r3, [sp, #16]
  28:  e58d3018     str   r3, [sp, #24]
  2c:  e59d2010     ldr   r2, [sp, #16]
  30:  e59d3014     ldr   r3, [sp, #20]
  34:  e0823003     add   r3, r2, r3
  38:  e58d3010     str   r3, [sp, #16]
  3c:  e59d3018     ldr   r3, [sp, #24]
  40:  e58d3014     str   r3, [sp, #20]
  44:  e59d300c     ldr   r3, [sp, #12]
  48:  e2833001     add   r3, r3, #1
  4c:  e58d300c     str   r3, [sp, #12]
```

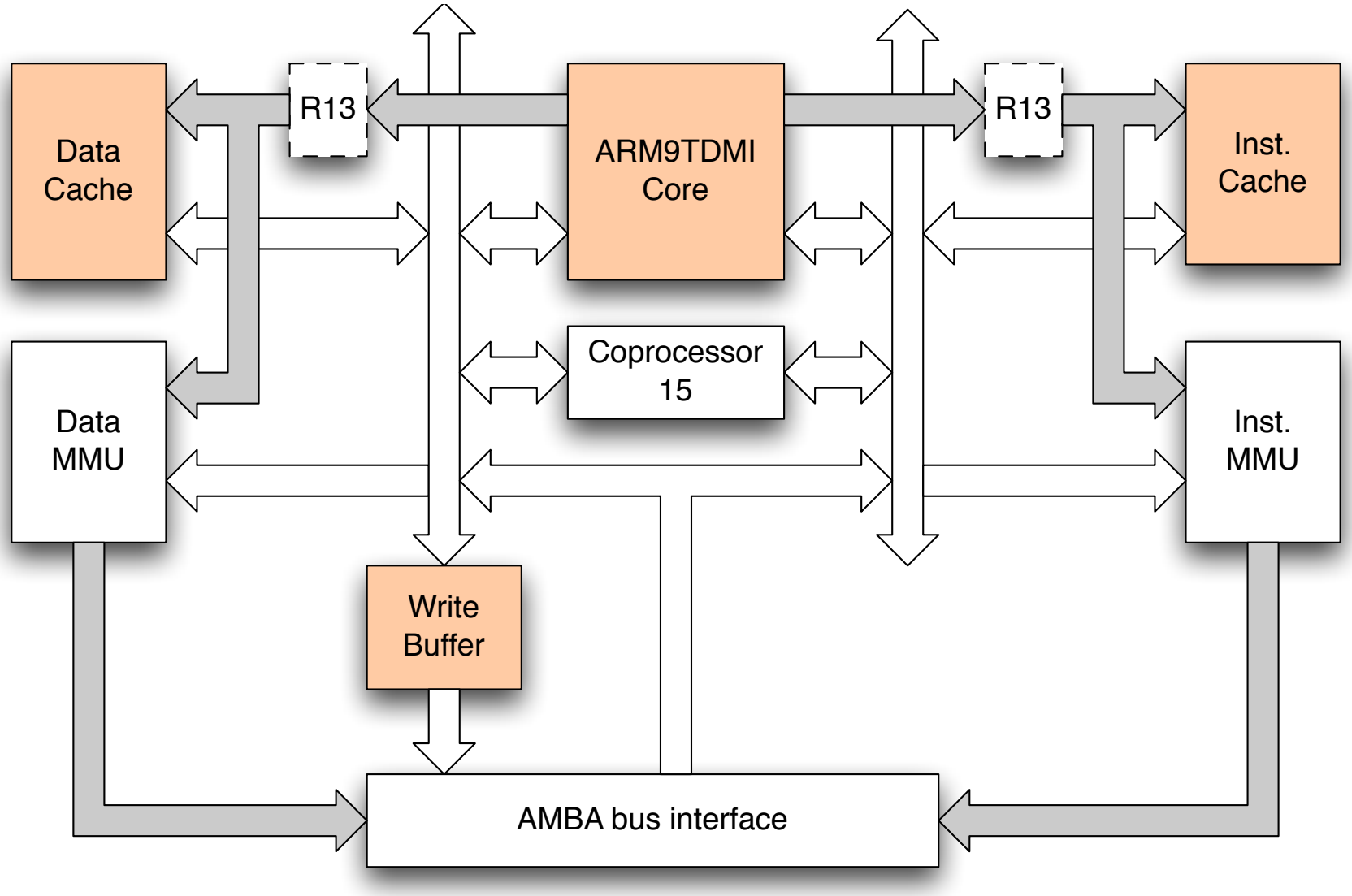


The Fibonacci Program

```
 54:  e59d3004    ldr   r3, [sp, #4]
 58:  e1520003    cmp   r2, r3
 5c:  dafffff0    ble   24 <fib+0x24>
 60:  e59d3010    ldr   r3, [sp, #16]
 64:  e58d301c    str   r3, [sp, #28]
 68:  e59d301c    ldr   r3, [sp, #28]
 6c:  e1a00003    mov   r0, r3
 70:  e28dd020    add   sp, sp, #32
 74:  e12fff1e    bx    lr

00000078 <main>:
 78:  e52de004    push  {lr}
 7c:  e24dd00c    sub   sp, sp, #12
 80:  e3a03f4b    mov   r3, #300
 84:  e58d3004    str   r3, [sp, #4]
 88:  e59d0004    ldr   r0, [sp, #4]
 8c:  ebffffdb    bl    0 <fib>
 90:  e1a03000    mov   r3, r0
 94:  e1a00003    mov   r0, r3
 98:  e28dd00c    add   sp, sp, #12
 9c:  e49de004    pop   {lr}
 a0:  e12fff1e    bx    lr
```

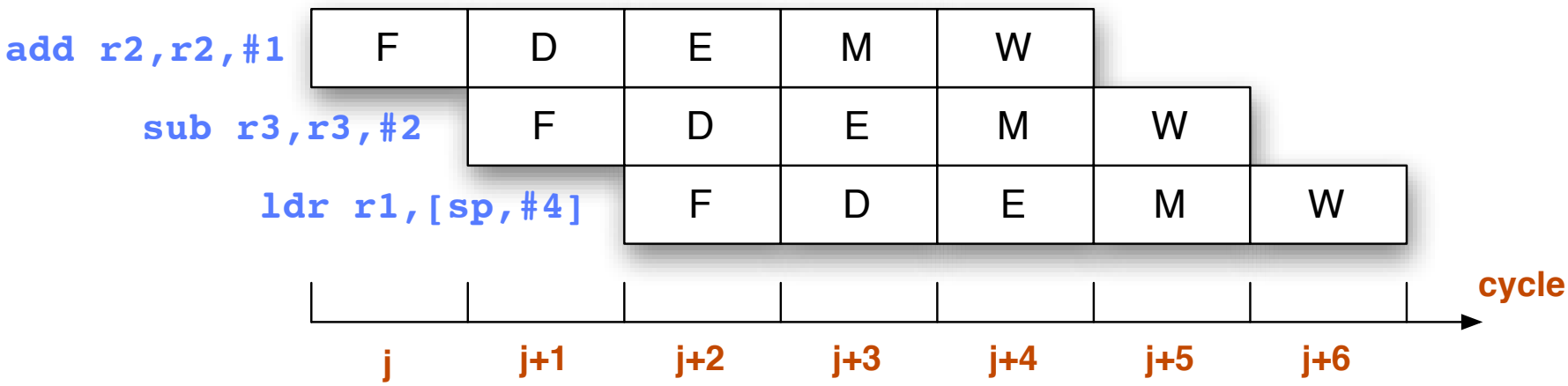
Target Architecture: ARM9xx family - ARM920T

- RISC processor, 16 registers, memory load/store and multiple ldr/str
- Data and Instruction Caches



Pipeline of the ARM920T

Pipelining = split execution of instructions into simple stages

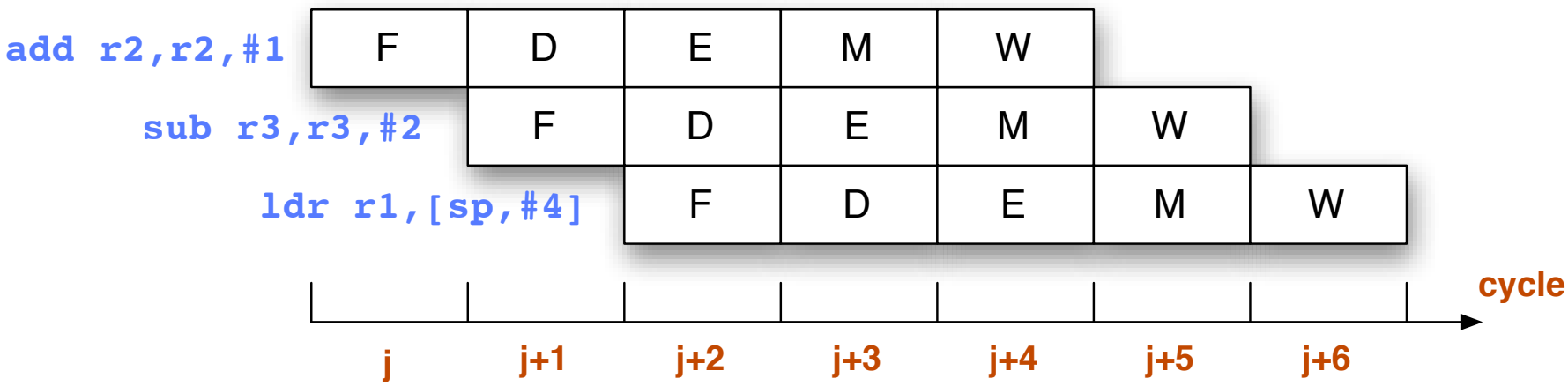


Concurrent execution of stages : on average one cycle per instruction

. . . but sometimes pipeline stalls

Pipeline of the ARM920T

Pipelining = split execution of instructions into simple stages

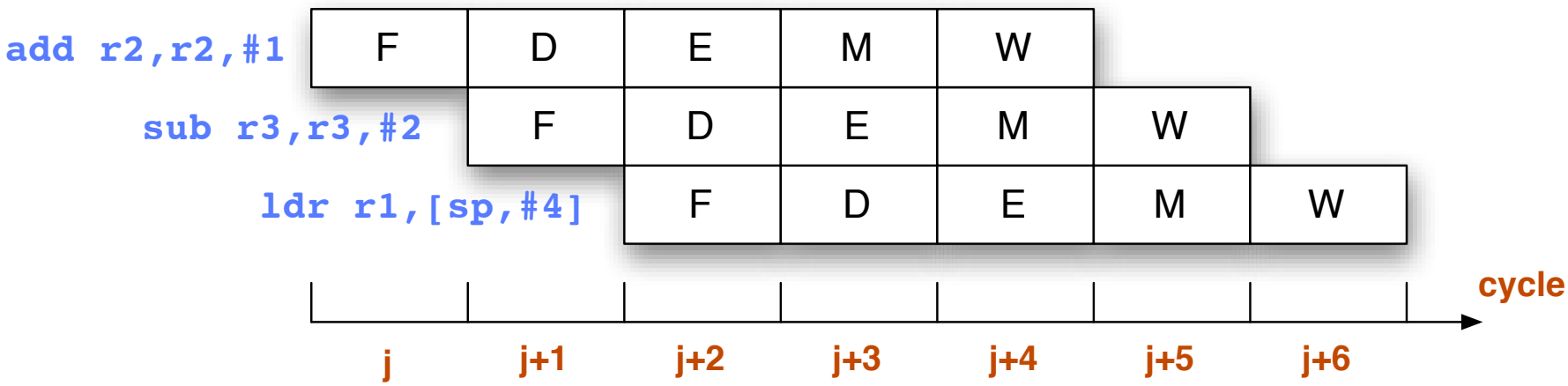


Concurrent execution of stages : on average one cycle per instruction

. . . but sometimes pipeline stalls

Pipeline of the ARM920T

Pipelining = split execution of instructions into simple stages

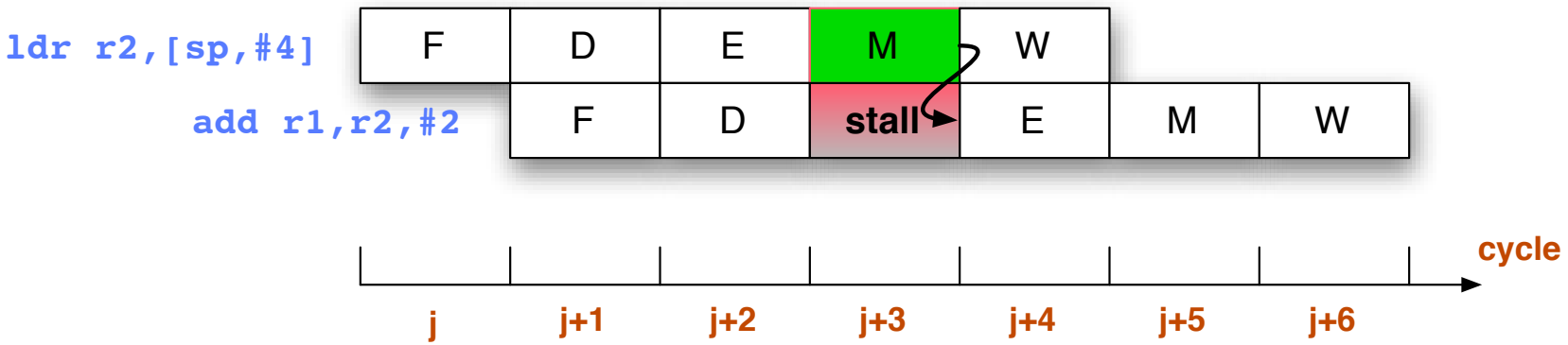


Concurrent execution of stages : on average one cycle per instruction

. . . but sometimes pipeline stalls

Pipeline Stalls

- Data dependences between instructions



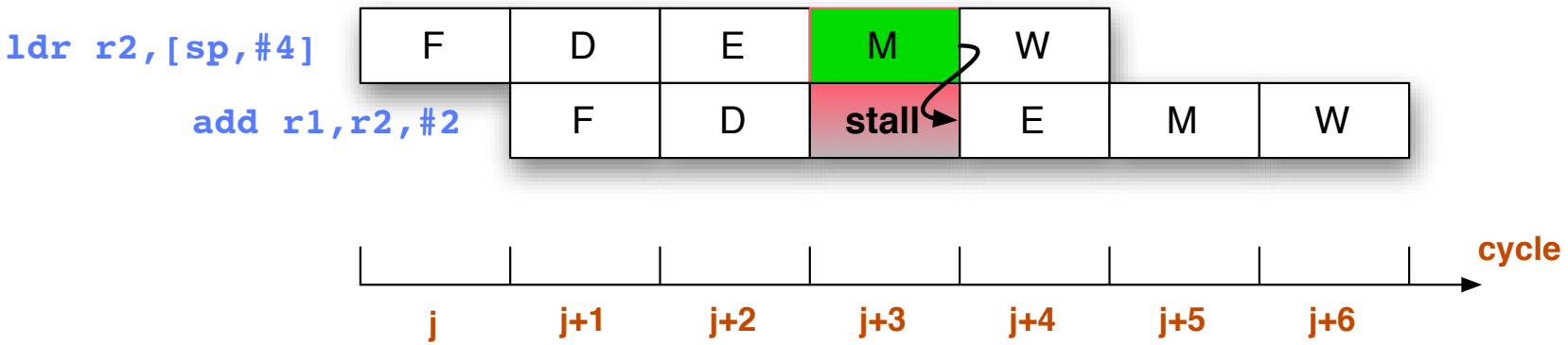
- Next instruction is a target of a "branch" instruction

Summary:

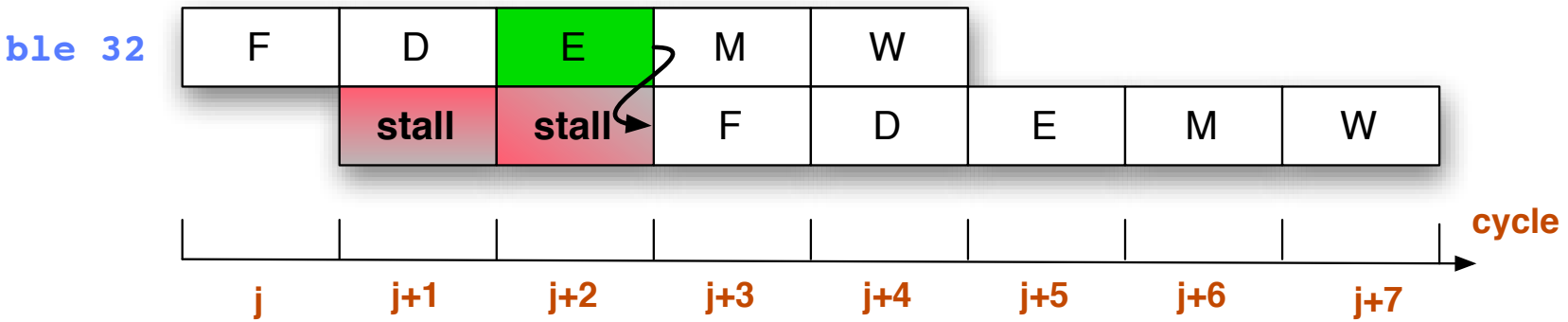
- on ARM9xxx: no branch prediction
- because of stalls, optimal flow (1 instruction/cycle) can be slowed down

Pipeline Stalls

- Data dependences between instructions



- Next instruction is a target of a "branch" instruction

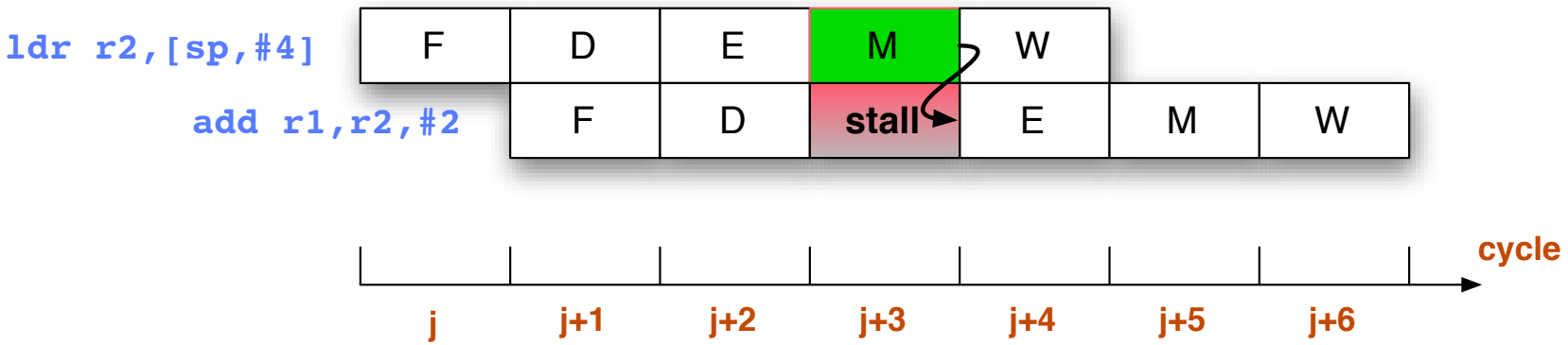


Summary:

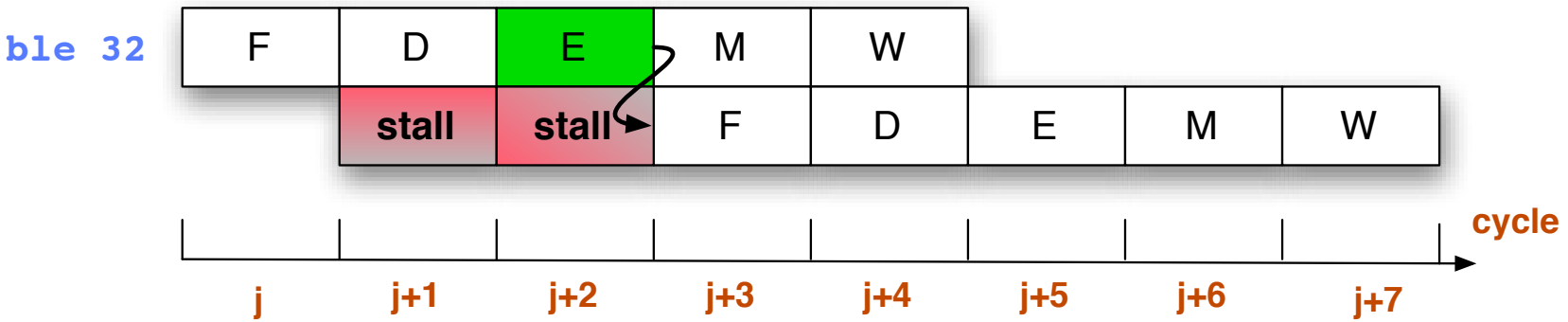
- on ARM9xxx: no branch prediction
- because of stalls, optimal flow (1 instruction/cycle) can be slowed down

Pipeline Stalls

- Data dependences between instructions



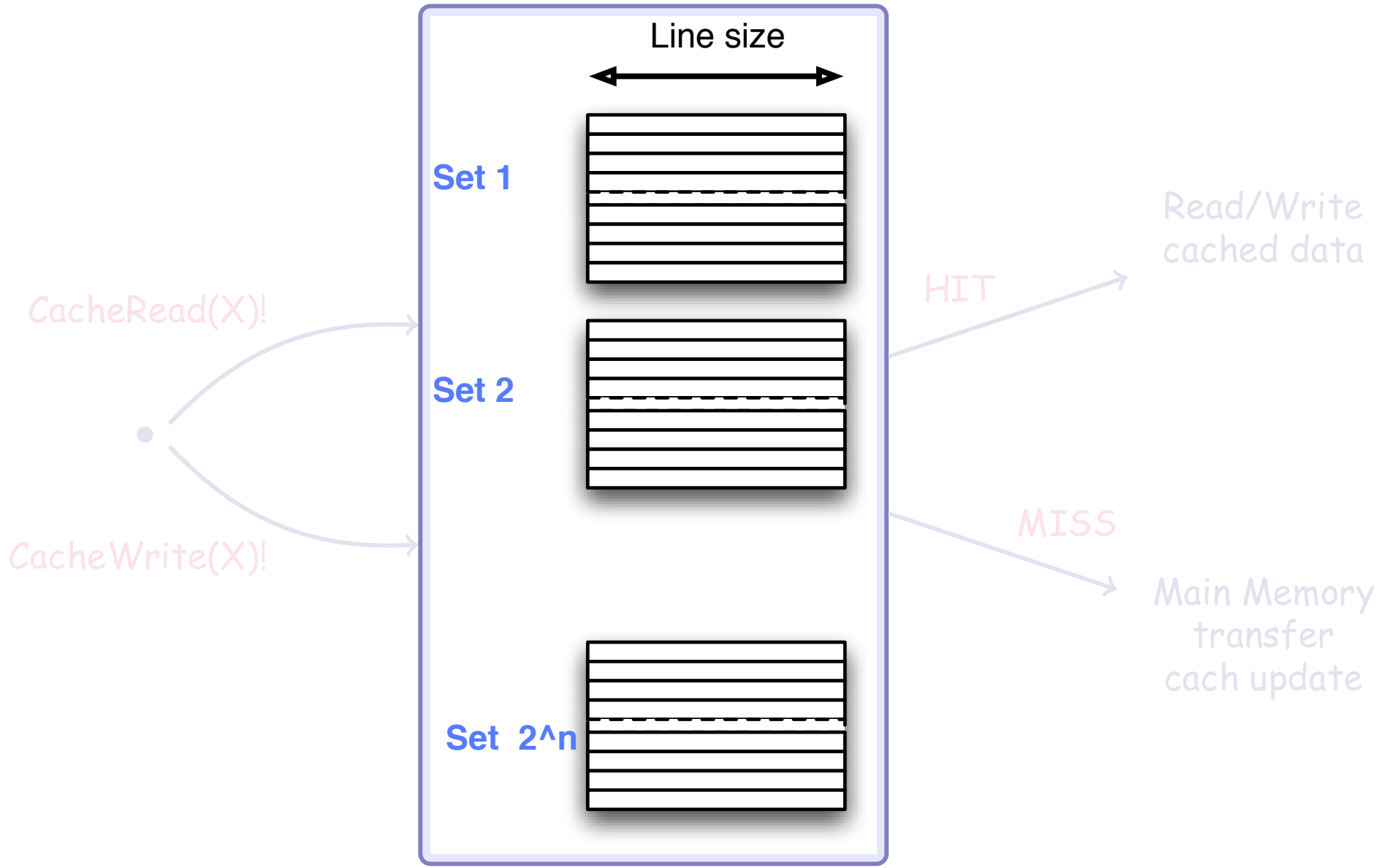
- Next instruction is a target of a "branch" instruction



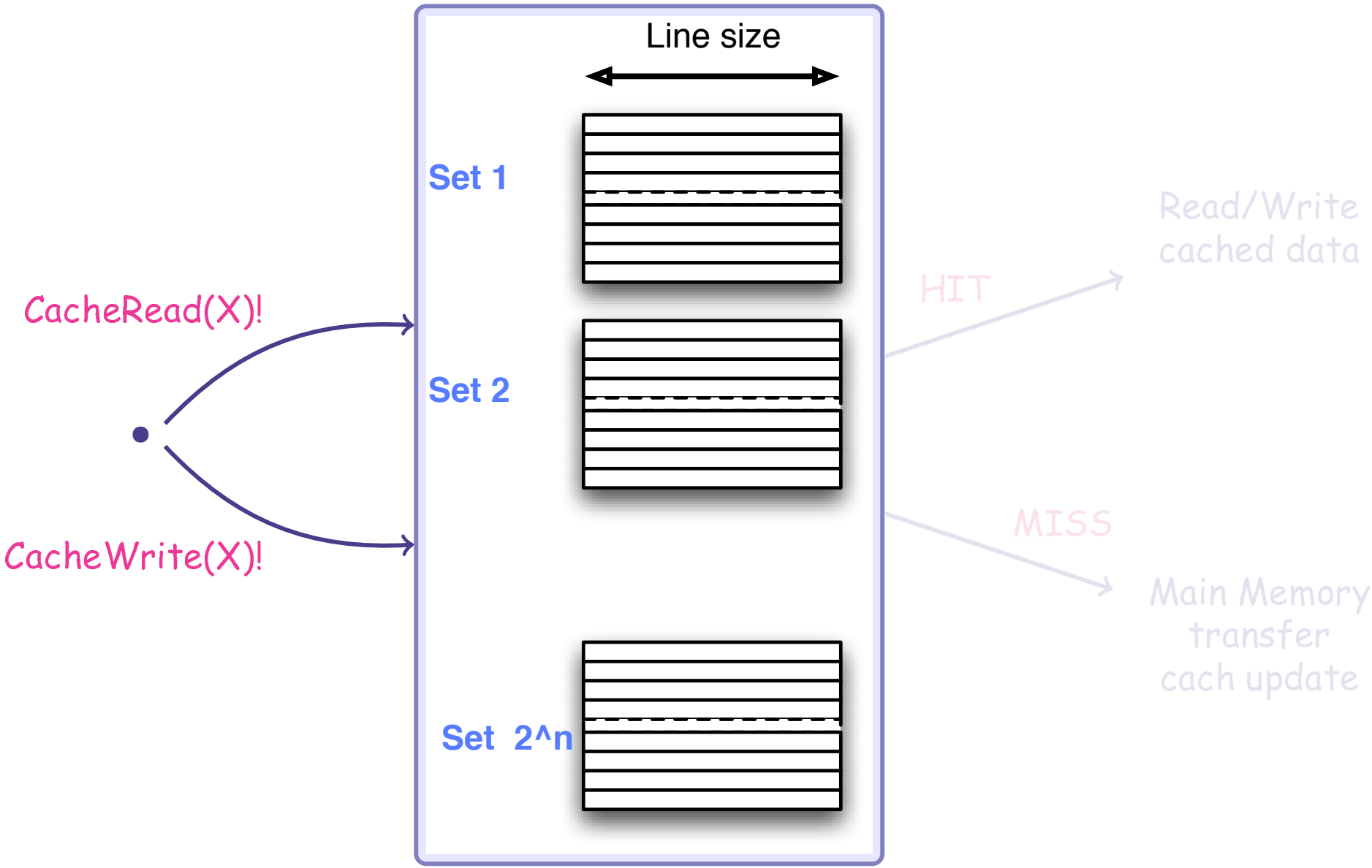
Summary:

- on ARM9xxx: no branch prediction
- because of stalls, optimal flow (1 instruction/cycle) can be slowed down

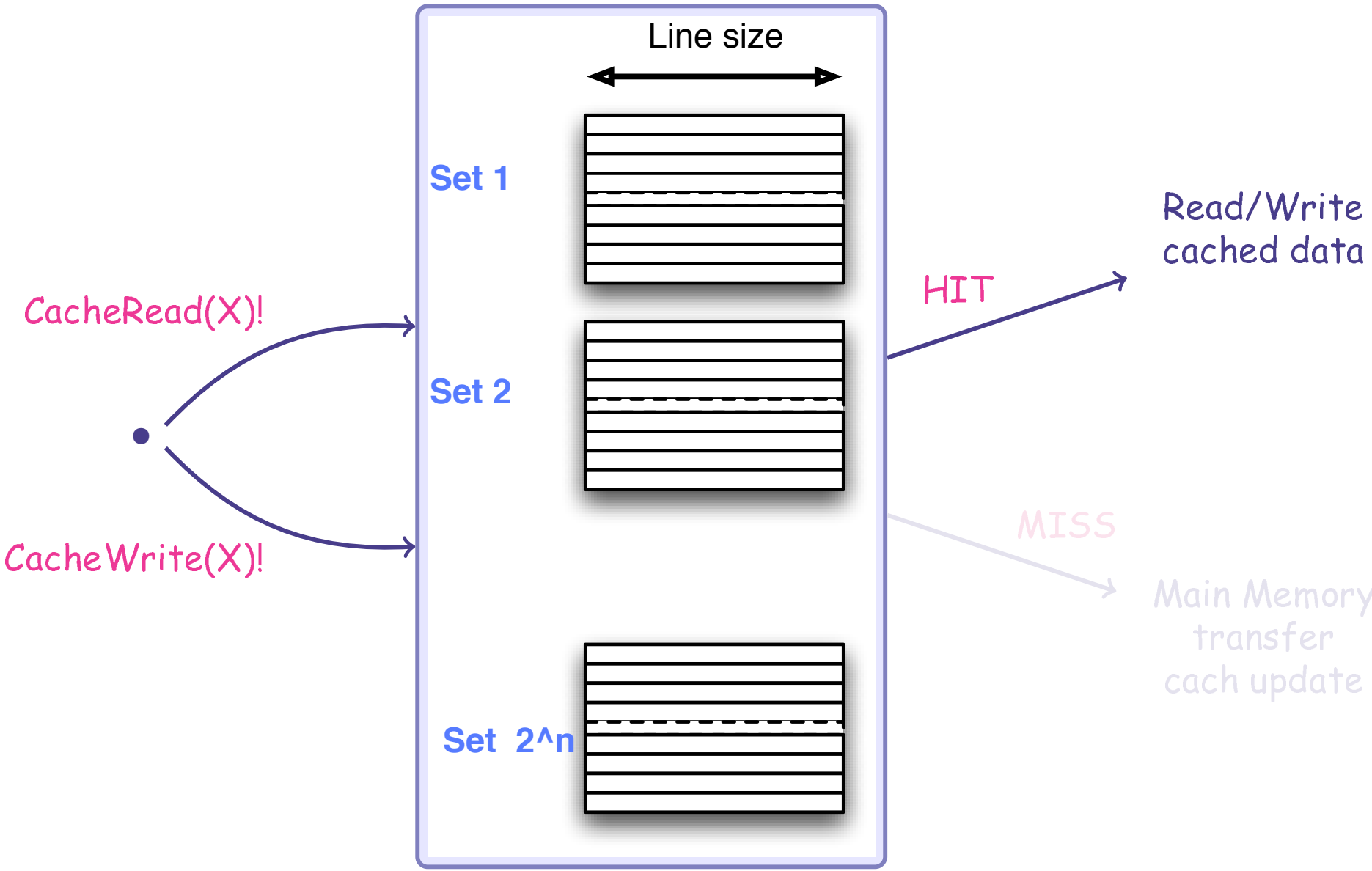
Caches


Line size
Set 1
Set 2
Set 2^n
CacheRead(X)!
CacheWrite(X)!
HIT
MISS
Read/Write
cached data
Main Memory
transfer
cach update

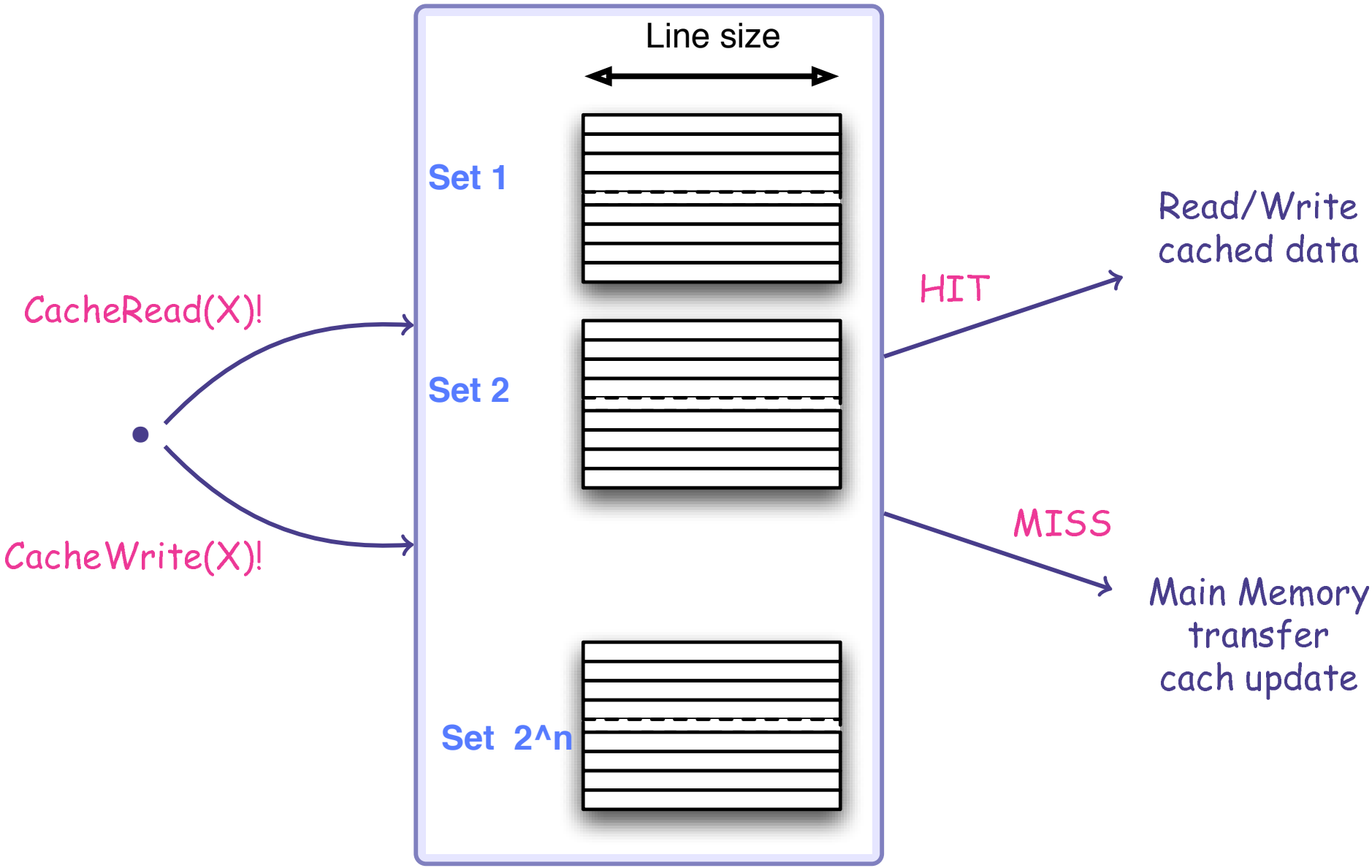
Caches


Line size
Set 1
Set 2
Set 2^n
CacheRead(X)!
CacheWrite(X)!
HIT
MISS
Read/Write
cached data
Main Memory
transfer
cach update

Caches

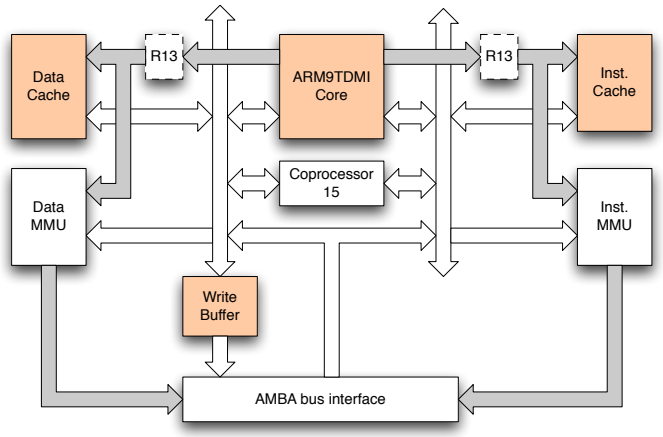

Line size
Set 1
Set 2
Set 2^n
CacheRead(X)!
CacheWrite(X)!
HIT
MISS
Read/Write
cached data
Main Memory
transfer
cach update

Caches


Line size
Set 1
Set 2
Set 2^n
CacheRead(X)!
CacheWrite(X)!
HIT
MISS
Read/Write
cached data
Main Memory
transfer
cach update

What is really needed to compute WCET ?

```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  daffffff0   ble   24 <fib+0x24>
```

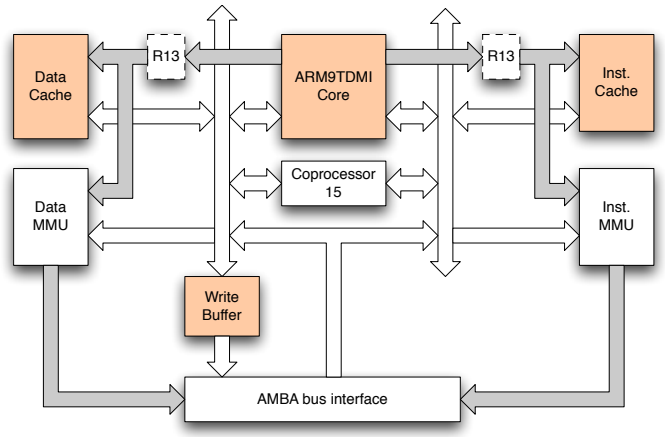


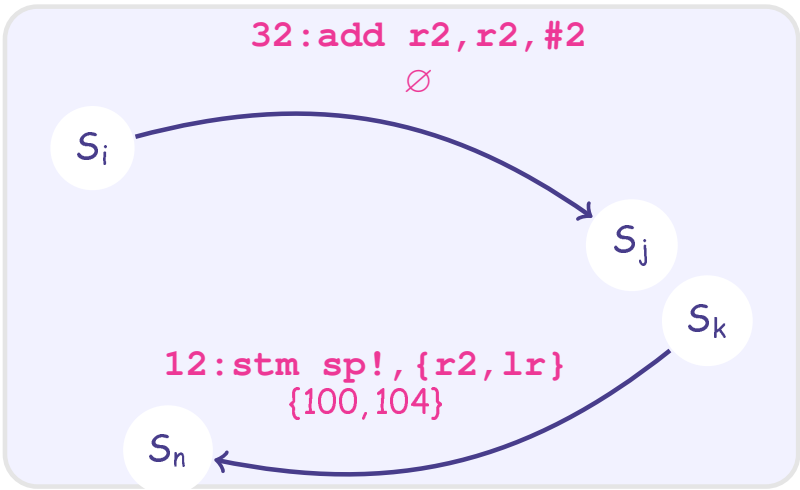
$\mathcal{L}(P) \subseteq \Sigma^*$

$\Sigma^* \longrightarrow \mathbb{N}$

What is really needed to compute WCET ?

```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  daffffff    ble   24 <fib+0x24>
```



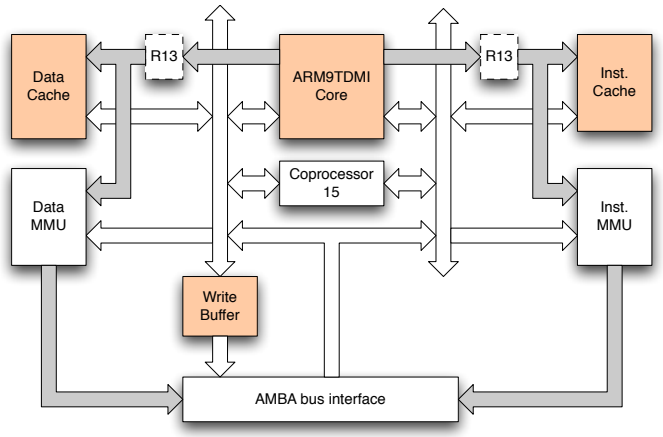


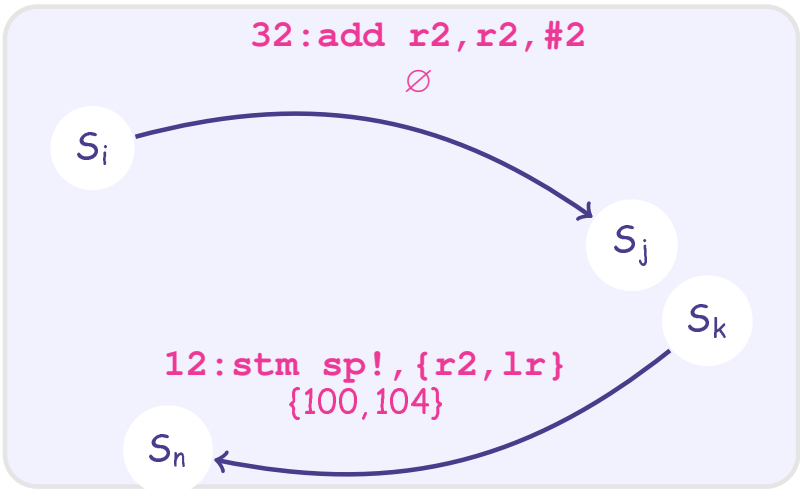
$\mathcal{L}(P) \subseteq \Sigma^*$

$\Sigma^* \longrightarrow \mathbb{N}$

What is really needed to compute WCET ?

```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  daffffff    ble   24 <fib+0x24>
```



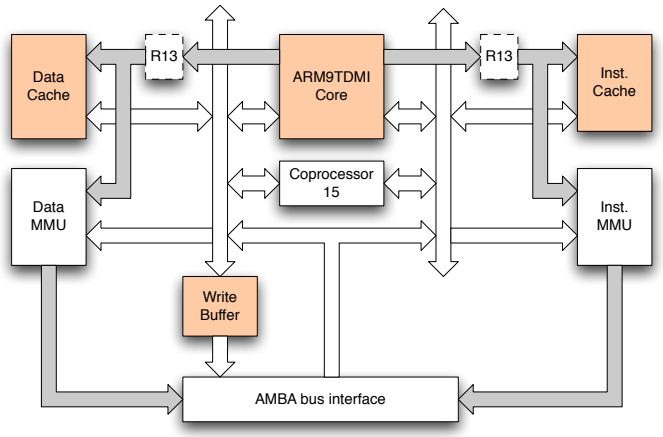


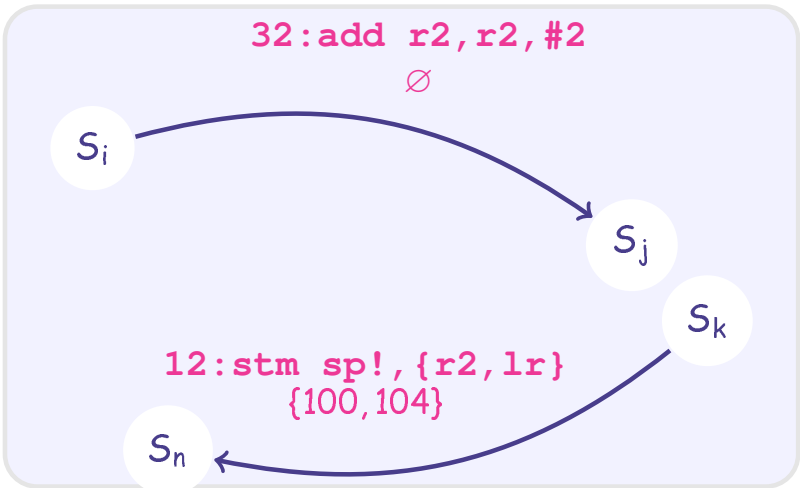
$\mathcal{L}(P) \subseteq \Sigma^*$

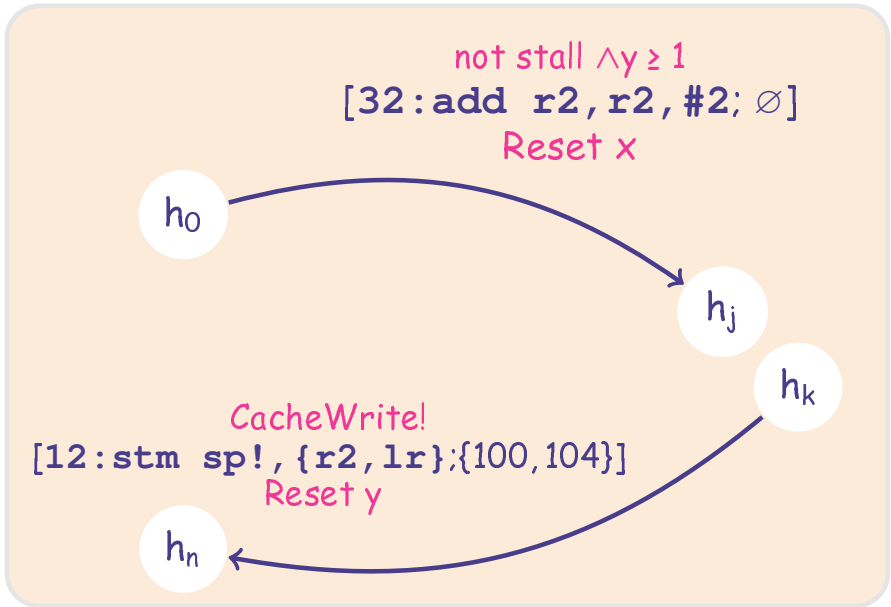
$\Sigma^* \longrightarrow \mathbb{N}$

What is really needed to compute WCET ?

```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  daffffff    ble   24 <fib+0x24>
```





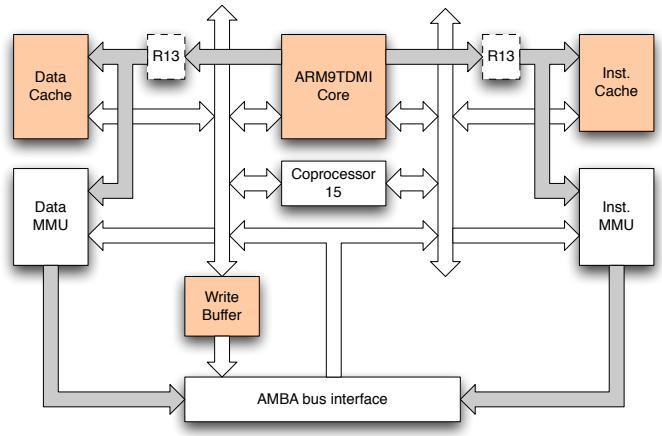


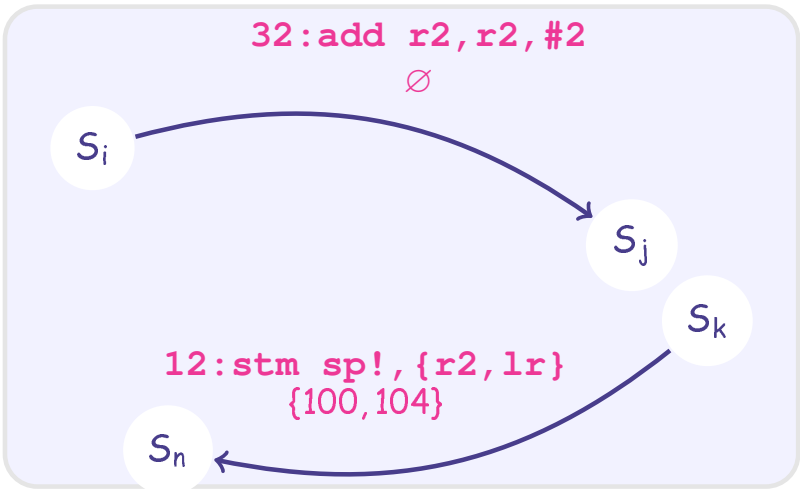
$\mathcal{L}(P) \subseteq \Sigma^*$

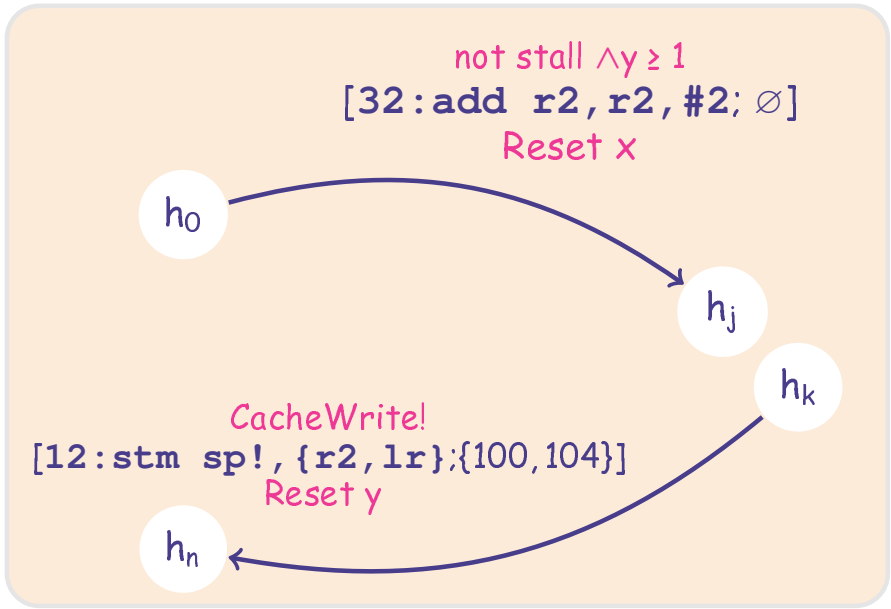
$\Sigma^* \longrightarrow \mathbb{N}$

What is really needed to compute WCET ?

```
10:  e3a03000    mov   r3, #0
14:  e58d3014    str   r3, [sp, #20]
18:  e3a03002    mov   r3, #2
1c:  e58d300c    str   r3, [sp, #12]
20:  ea00000a    b     50 <fib+0x50>
24:  e59d3010    ldr   r3, [sp, #16]
28:  e58d3018    str   r3, [sp, #24]
2c:  e59d2010    ldr   r2, [sp, #16]
30:  e59d3014    ldr   r3, [sp, #20]
34:  e0823003    add   r3, r2, r3
38:  e58d3010    str   r3, [sp, #16]
3c:  e59d3018    ldr   r3, [sp, #24]
40:  e58d3014    str   r3, [sp, #20]
44:  e59d300c    ldr   r3, [sp, #12]
48:  e2833001    add   r3, r3, #1
4c:  e58d300c    str   r3, [sp, #12]
50:  e59d200c    ldr   r2, [sp, #12]
54:  e59d3004    ldr   r3, [sp, #4]
58:  e1520003    cmp   r2, r3
5c:  daffffff    ble   24 <fib+0x24>
```



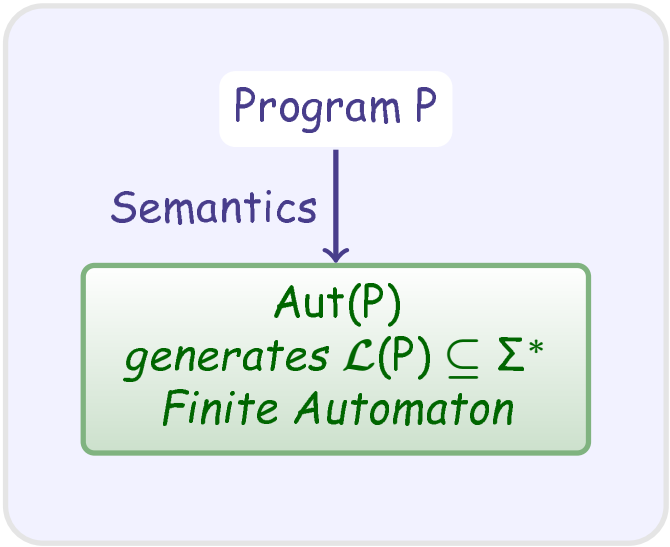




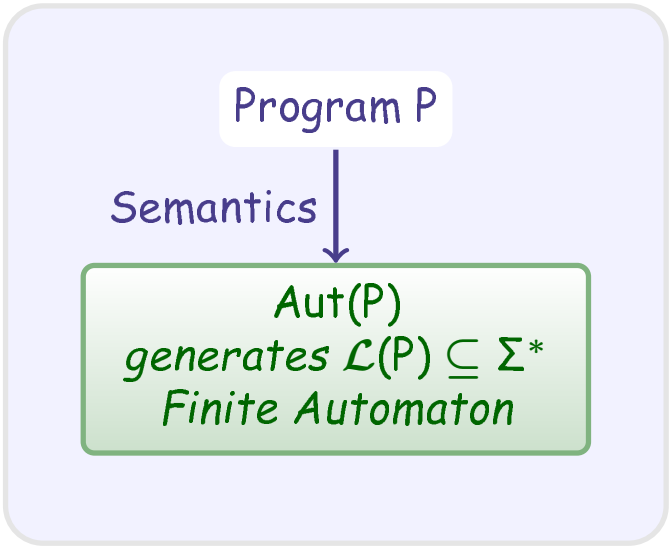
$\mathcal{L}(P) \subseteq \Sigma^*$

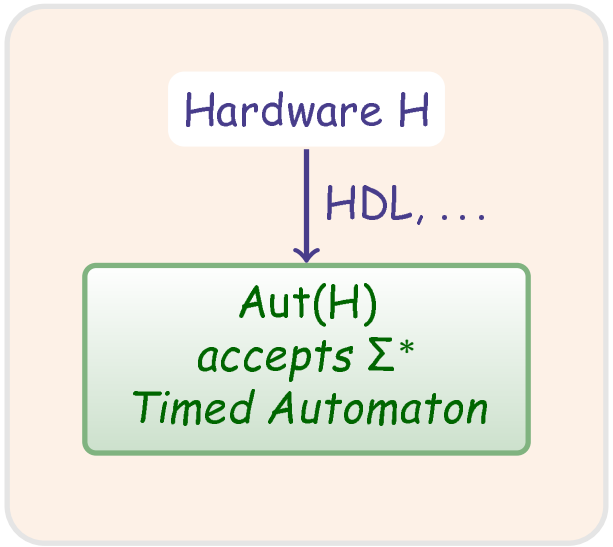
$\Sigma^* \longrightarrow \mathbb{N}$

Modular Computation of WCET

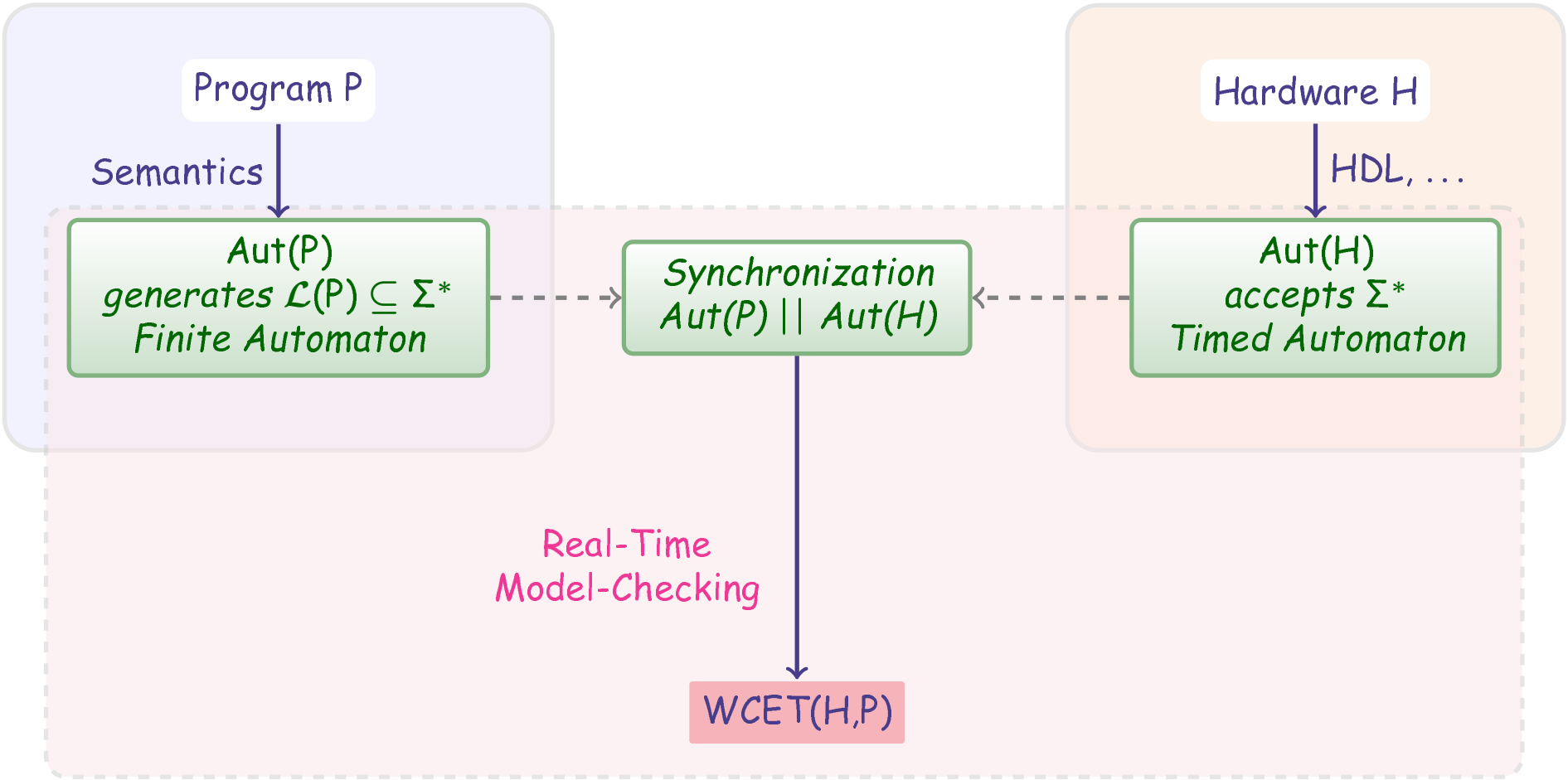


Modular Computation of WCET





Modular Computation of WCET



WCET-Equivalent Reduced Program

- Two runs of P can generate the same word in $\mathcal{L}(P)$
  e.g., Fibonacci with initial values $u_0 = 0, u_1 = 1$ and $u_0 = 2, u_1 = 3$
- state of Aut(P): 16 32-bit registers, stack, status bits
  size of a state of Aut(P): $16 \times 32 + |\text{stack}| \times 32 + 4$
- WCET depends on $\mathcal{L}(P)$

$$\text{if } \mathcal{L}(P') = \mathcal{L}(P) \text{ then } \text{WCET}(H, P) = \text{WCET}(H, P')$$

WCET-equivalent Program
P′ and P are WCET-equivalent iff $\mathcal{L}(P') = \mathcal{L}(P)$.

Compute a reduced WCET-equivalent P′ using Program Slicing

[Weiser, 1984] Mark Weiser.
Program slicing.
IEEE Trans. Software Eng., 10(4):352–357, 1984.

WCET-Equivalent Reduced Program

- Two runs of P can generate the same word in $\mathcal{L}(P)$
  e.g., Fibonacci with initial values $u_0 = 0, u_1 = 1$ and $u_0 = 2, u_1 = 3$
- state of Aut(P): 16 32-bit registers, stack, status bits
  size of a state of Aut(P): $16 \times 32 + |\text{stack}| \times 32 + 4$
- WCET depends on $\mathcal{L}(P)$

$$\text{if } \mathcal{L}(P') = \mathcal{L}(P) \text{ then } \text{WCET}(H, P) = \text{WCET}(H, P')$$

WCET-equivalent Program
P′ and P are WCET-equivalent iff $\mathcal{L}(P') = \mathcal{L}(P)$.

Compute a reduced WCET-equivalent P′ using Program Slicing

[Weiser, 1984] Mark Weiser.
Program slicing.
IEEE Trans. Software Eng., 10(4):352–357, 1984.

WCET-Equivalent Reduced Program

- Two runs of P can generate the same word in $\mathcal{L}(P)$
  e.g., Fibonacci with initial values $u_0 = 0, u_1 = 1$ and $u_0 = 2, u_1 = 3$
- state of Aut(P): 16 32-bit registers, stack, status bits
  size of a state of Aut(P): $16 \times 32 + |\text{stack}| \times 32 + 4$
- WCET depends on $\mathcal{L}(P)$

$$\text{if } \mathcal{L}(P') = \mathcal{L}(P) \text{ then } \text{WCET}(H, P) = \text{WCET}(H, P')$$

WCET-equivalent Program
P' and P are WCET-equivalent iff $\mathcal{L}(P') = \mathcal{L}(P)$.

Compute a reduced WCET-equivalent P' using Program Slicing

[Weiser, 1984] Mark Weiser.
Program slicing.
IEEE Trans. Software Eng., 10(4):352–357, 1984.

WCET-Equivalent Reduced Program

- Two runs of P can generate the same word in $\mathcal{L}(P)$
  e.g., Fibonacci with initial values $u_0 = 0, u_1 = 1$ and $u_0 = 2, u_1 = 3$
- state of Aut(P): 16 32-bit registers, stack, status bits
  size of a state of Aut(P): $16 \times 32 + |\text{stack}| \times 32 + 4$
- WCET depends on $\mathcal{L}(P)$

$$\text{if } \mathcal{L}(P') = \mathcal{L}(P) \text{ then } \text{WCET}(H, P) = \text{WCET}(H, P')$$

WCET-equivalent Program
P′ and P are WCET-equivalent iff $\mathcal{L}(P') = \mathcal{L}(P)$.

Compute a reduced WCET-equivalent P′ using Program Slicing

[Weiser, 1984] Mark Weiser.
Program slicing.
IEEE Trans. Software Eng., 10(4):352–357, 1984.

WCET-Equivalent Reduced Program

- Two runs of P can generate the same word in $\mathcal{L}(P)$
  e.g., Fibonacci with initial values $u_0 = 0, u_1 = 1$ and $u_0 = 2, u_1 = 3$
- state of Aut(P): 16 32-bit registers, stack, status bits
  size of a state of Aut(P): $16 \times 32 + |\text{stack}| \times 32 + 4$
- WCET depends on $\mathcal{L}(P)$

$$\text{if } \mathcal{L}(P') = \mathcal{L}(P) \text{ then } \text{WCET}(H, P) = \text{WCET}(H, P')$$

WCET-equivalent Program
P′ and P are WCET-equivalent iff $\mathcal{L}(P') = \mathcal{L}(P)$.

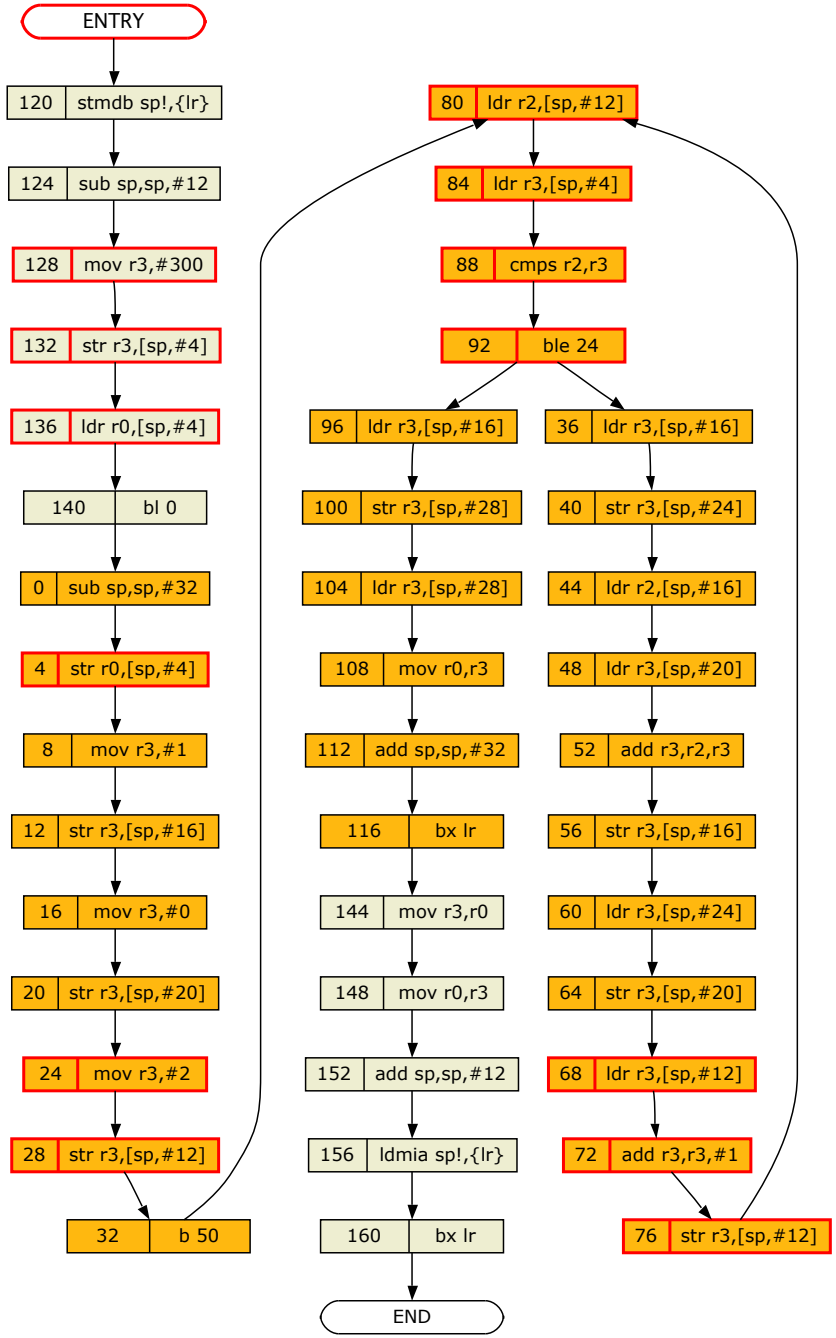
Compute a reduced WCET-equivalent P′ using Program Slicing

[Weiser, 1984] Mark Weiser.
Program slicing.
IEEE Trans. Software Eng., 10(4):352–357, 1984.

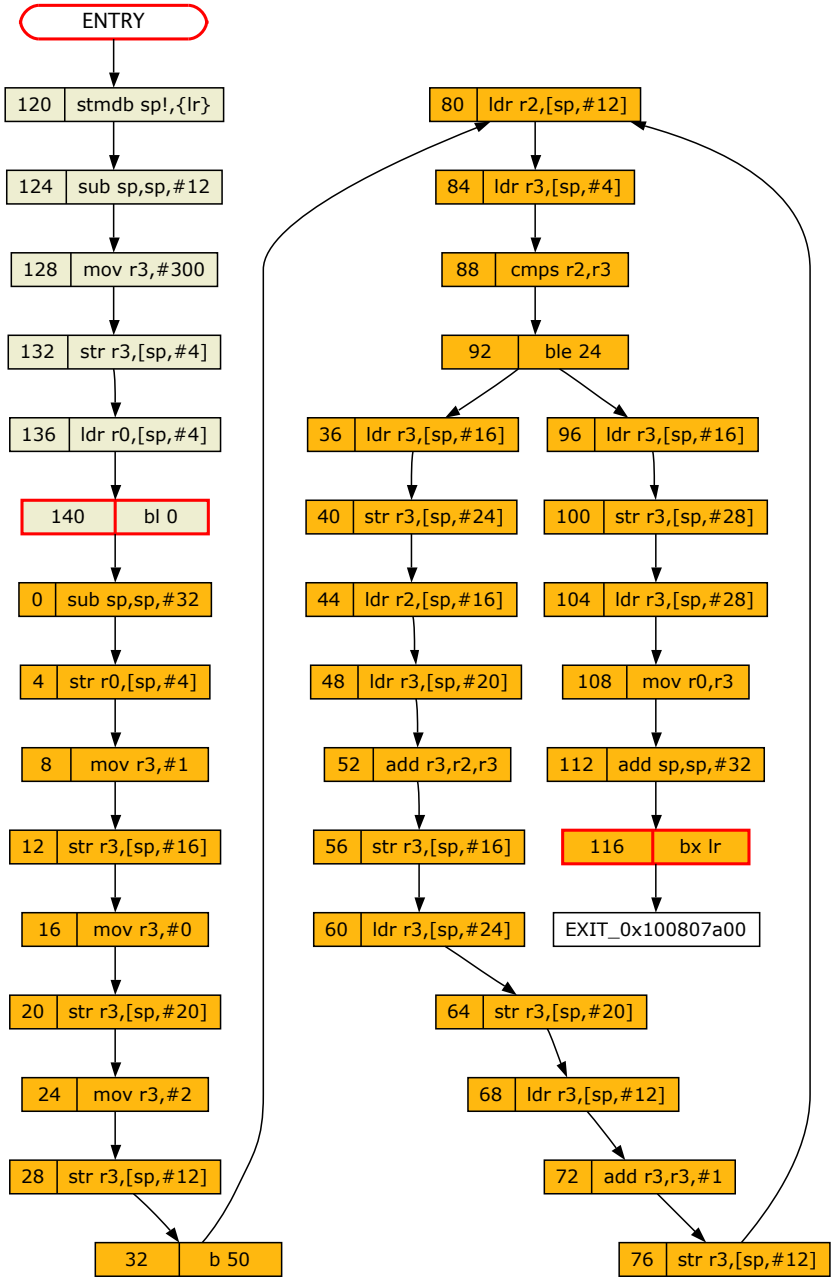
Sliced Fibonacci Program

{r0,r2,r3,stack_3020,stack_3028,stack_3052}

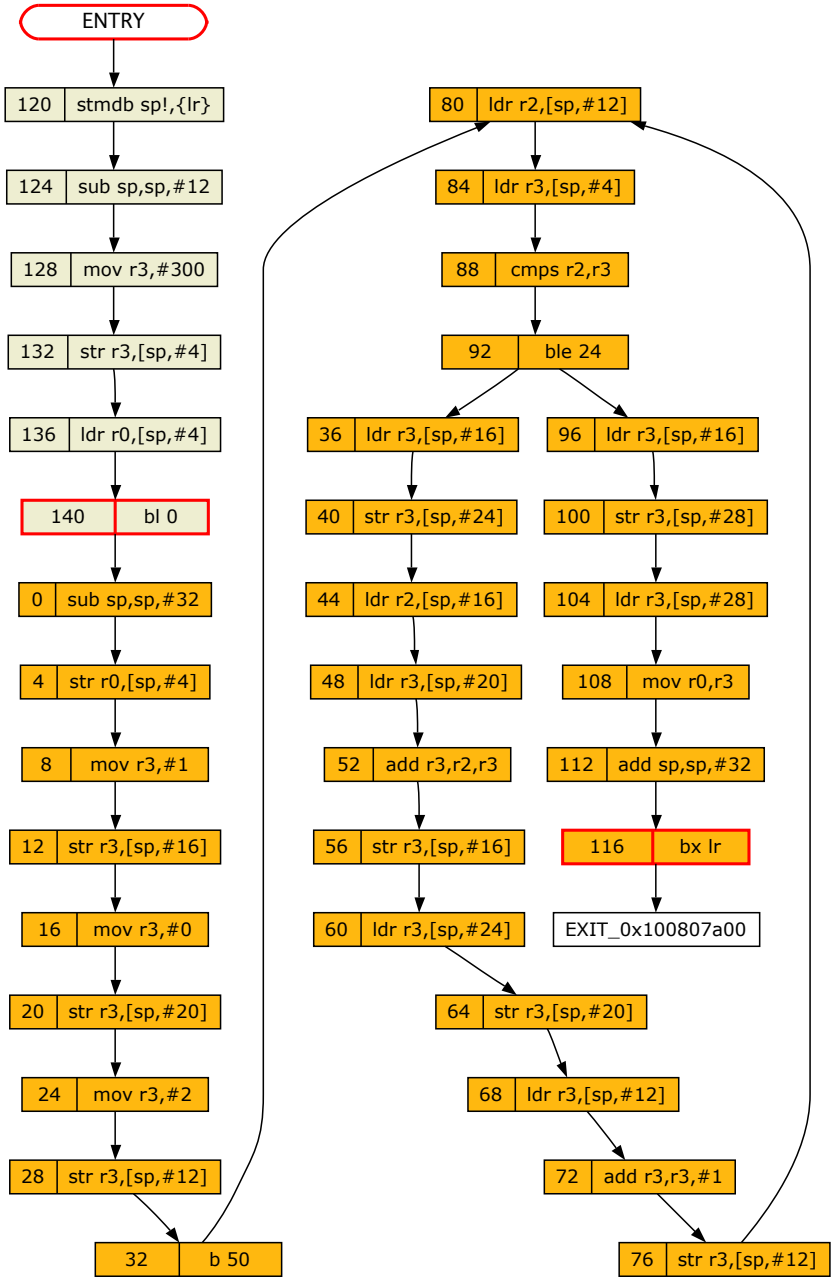
```
00000000 <fib>:
   0:  e24dd020     sub   sp, sp, #32
   4:  e58d0004     str   r0, [sp, #4]
   8:  e3a03001     mov   r3, #1
   c:  e58d3010     str   r3, [sp, #16]
  10:  e3a03000     mov   r3, #0
  14:  e58d3014     str   r3, [sp, #20]
  18:  e3a03002     mov   r3, #2          <-
  1c:  e58d300c     str   r3, [sp, #12]   <-
  20:  ea00000a     b     50 <fib+0x50>
  24:  e59d3010     ldr   r3, [sp, #16]
  28:  e58d3018     str   r3, [sp, #24]
  2c:  e59d2010     ldr   r2, [sp, #16]
  30:  e59d3014     ldr   r3, [sp, #20]
  34:  e0823003     add   r3, r2, r3
  38:  e58d3010     str   r3, [sp, #16]
  3c:  e59d3018     ldr   r3, [sp, #24]
  40:  e58d3014     str   r3, [sp, #20]
  44:  e59d300c     ldr   r3, [sp, #12]   <-
  48:  e2833001     add   r3, r3, #1      <-
  4c:  e58d300c     str   r3, [sp, #12]   <-
  50:  e59d200c     ldr   r2, [sp, #12]   <-
  54:  e59d3004     ldr   r3, [sp, #4]    <-
  58:  e1520003     cmp   r2, r3          <-
  5c:  dafffff0     ble   24 <fib+0x24>   <-
  60:  e59d3010     ldr   r3, [sp, #16]
  64:  e58d301c     str   r3, [sp, #28]
  68:  e59d301c     ldr   r3, [sp, #28]
  6c:  e1a00003     mov   r0, r3
  70:  e28dd020     add   sp, sp, #32
  74:  e12fff1e     bx    lr

00000078 <main>:
  78:  e52de004     push  {lr}
  7c:  e24dd00c     sub   sp, sp, #12
  80:  e3a03f4b     mov   r3, #300        <-
  84:  e58d3004     str   r3, [sp, #4]    <-
  88:  e59d0004     ldr   r0, [sp, #4]    <-
  8c:  ebffffdb     bl    0 <fib>
  90:  e1a03000     mov   r3, r0
  94:  e1a00003     mov   r0, r3
  98:  e28dd00c     add   sp, sp, #12
  9c:  e49de004     pop   {lr}
  a0:  e12fff1e     bx    lr
```

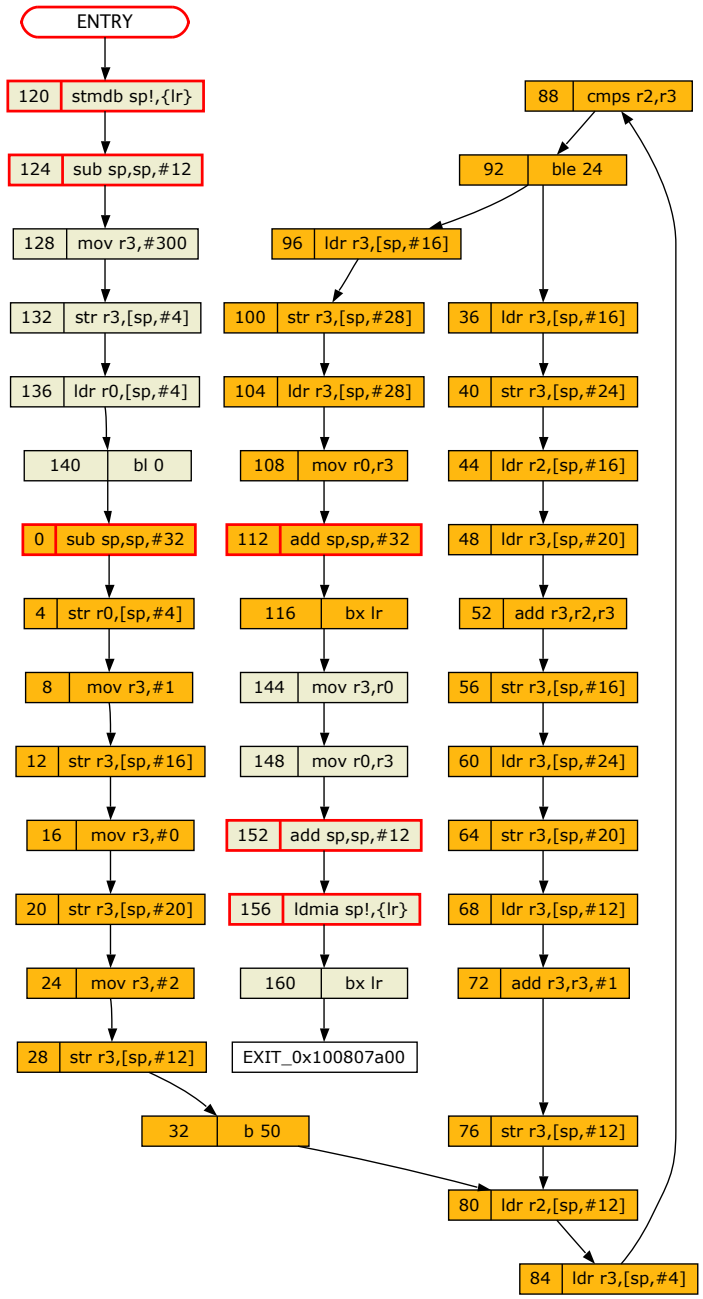


Automatic Computation of CFG using Program Slicing



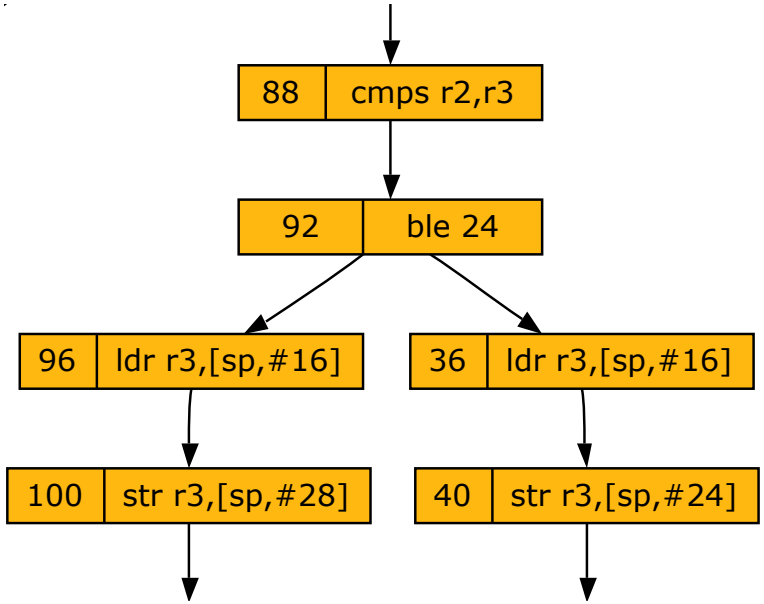
Automatic Computation of CFG using Program Slicing

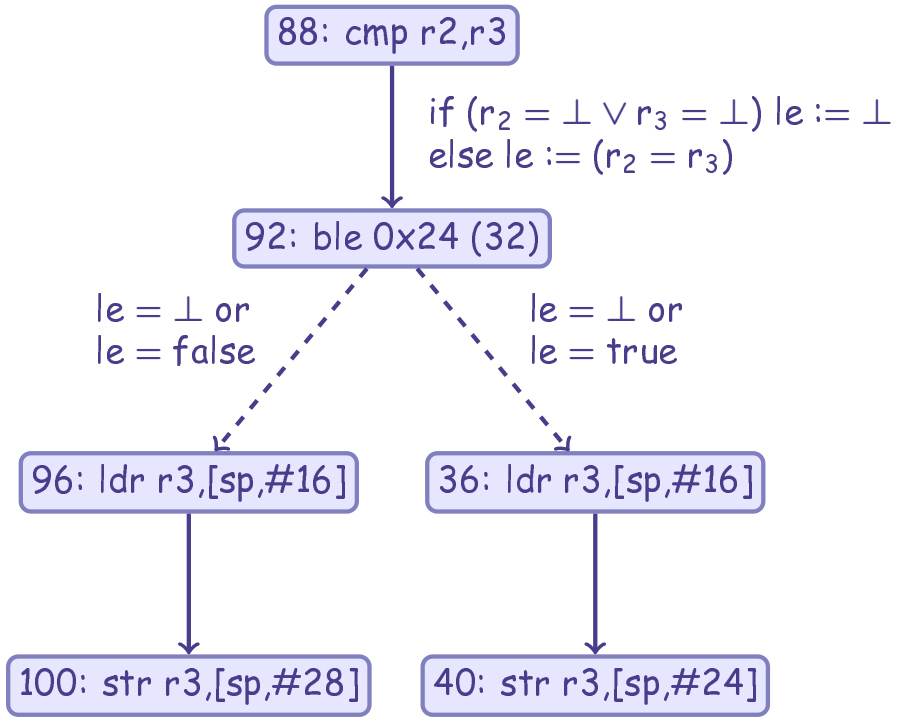




Handling Unknown Input Data

Input data are unknown: extended domain: $\mathcal{D}_\perp = \mathcal{D} \cup \{\perp\}$

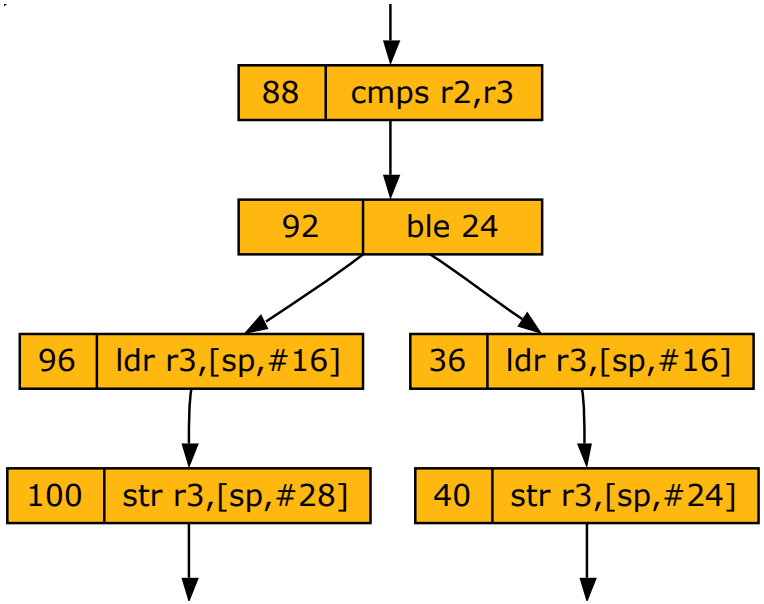


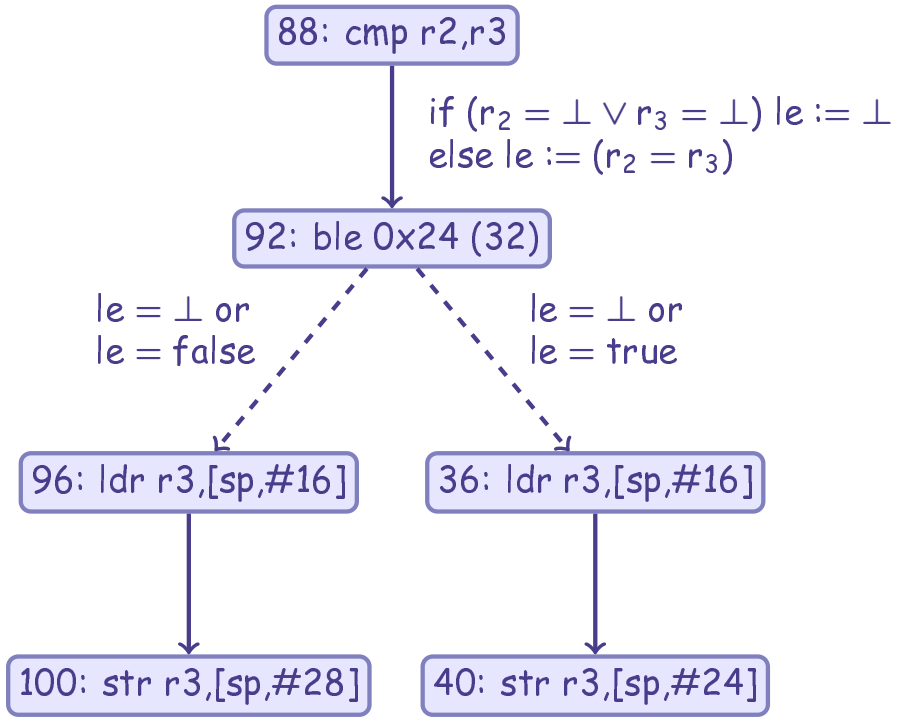


Game: program vs outcomes of comparisons

Handling Unknown Input Data

Input data are unknown: extended domain: $\mathcal{D}_\perp = \mathcal{D} \cup \{\perp\}$





Game: program vs outcomes of comparisons

Hardware Formal Models

Formal Models

- Timed Automata: automata extended with dense-time clocks

Hardware Specs ?

- Data sheets
- Incomplete or sketchy

Bad formal models $\Longrightarrow$ very bad WCET results

How can we build better models ?

- Find a specialist in computer architecture
- Design programs to stress particular features of the hardware
- Compare actual execution-times with computed execution-times
- Refine formal model

Hardware Formal Models

Formal Models

- Timed Automata: automata extended with dense-time clocks

Hardware Specs ?

- Data sheets
- Incomplete or sketchy

Bad formal models $\Longrightarrow$ very bad WCET results

How can we build better models ?

- Find a specialist in computer architecture
- Design programs to stress particular features of the hardware
- Compare actual execution-times with computed execution-times
- Refine formal model

Hardware Formal Models

Formal Models

- Timed Automata: automata extended with dense-time clocks

Hardware Specs ?

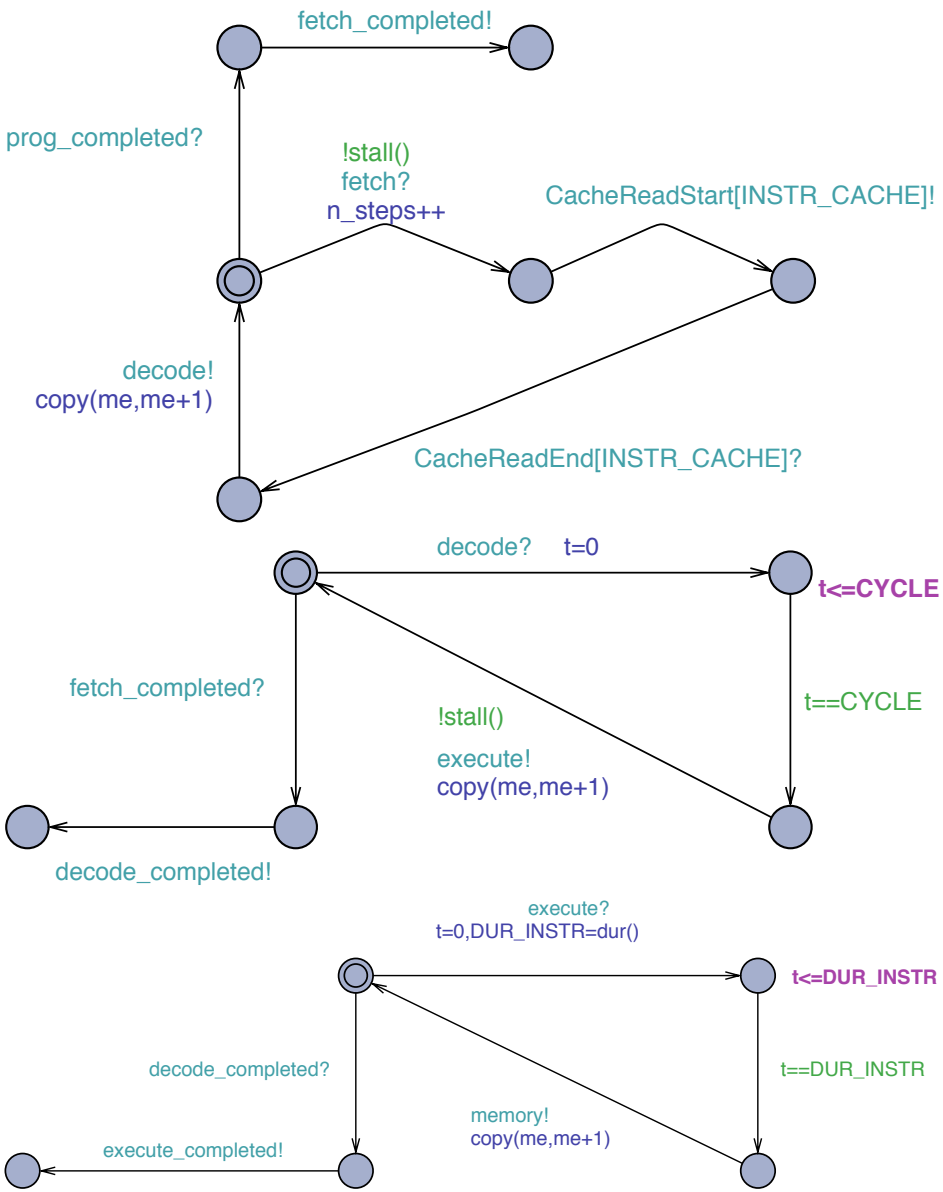
- Data sheets
- Incomplete or sketchy

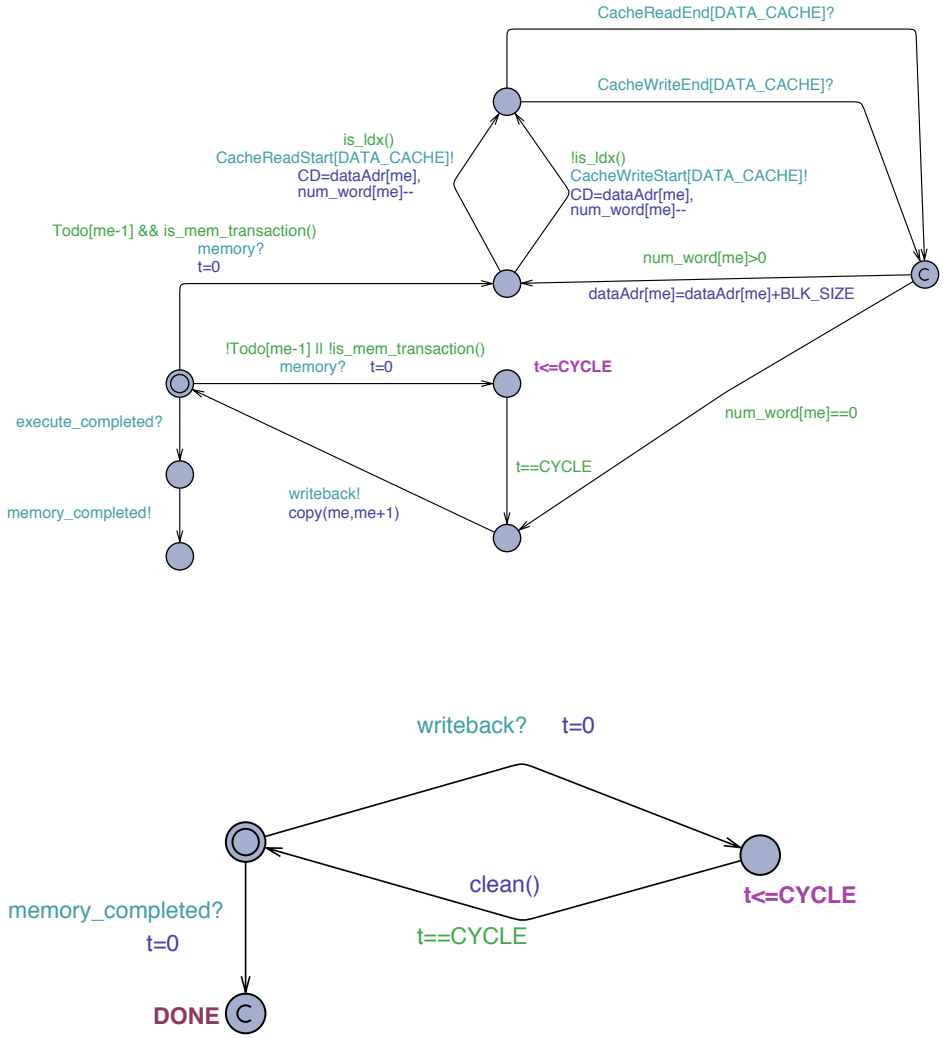
Bad formal models $\Longrightarrow$ very bad WCET results

How can we build better models ?

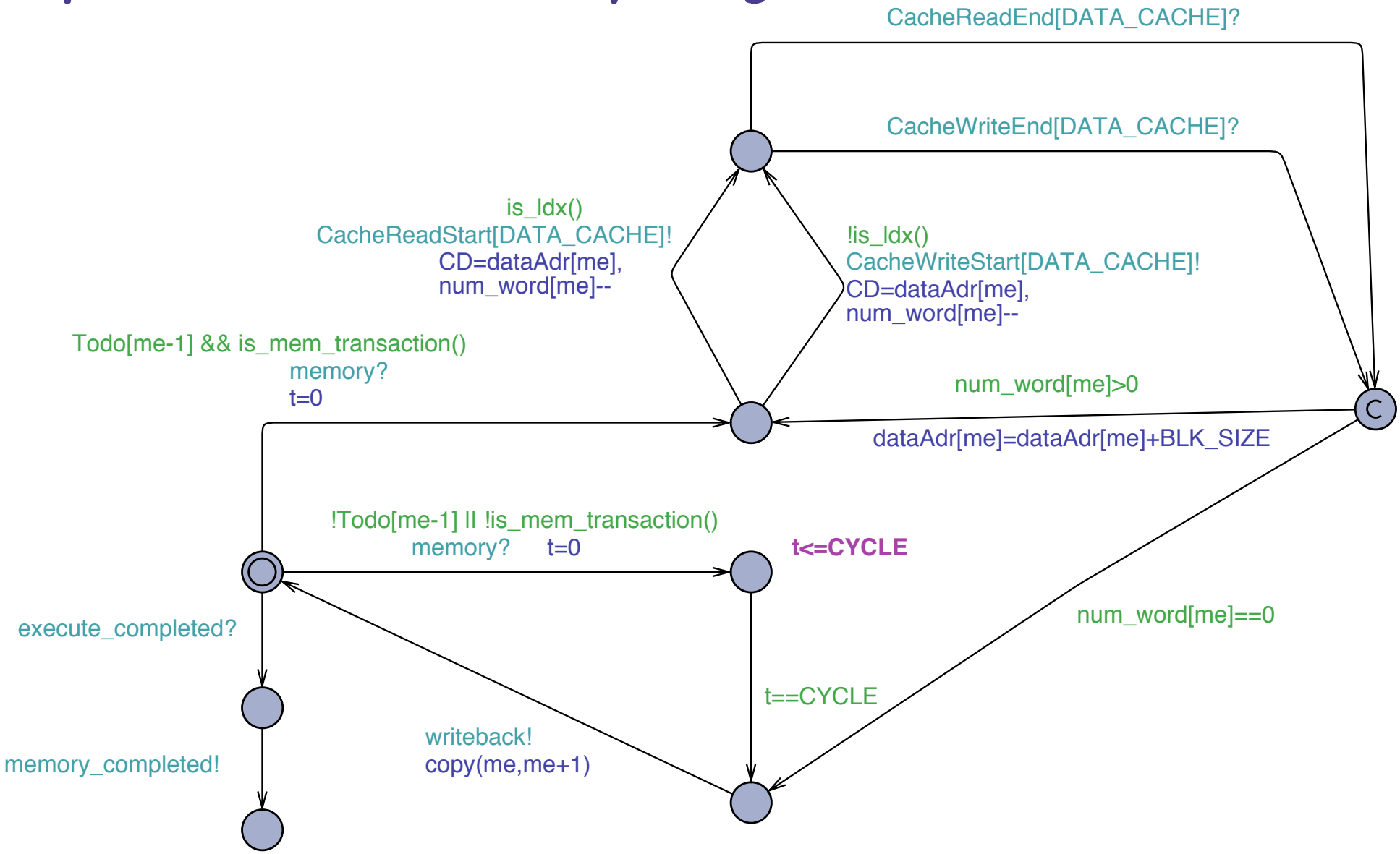
- Find a specialist in computer architecture
- Design programs to stress particular features of the hardware
- Compare actual execution-times with computed execution-times
- Refine formal model

Pipeline Model

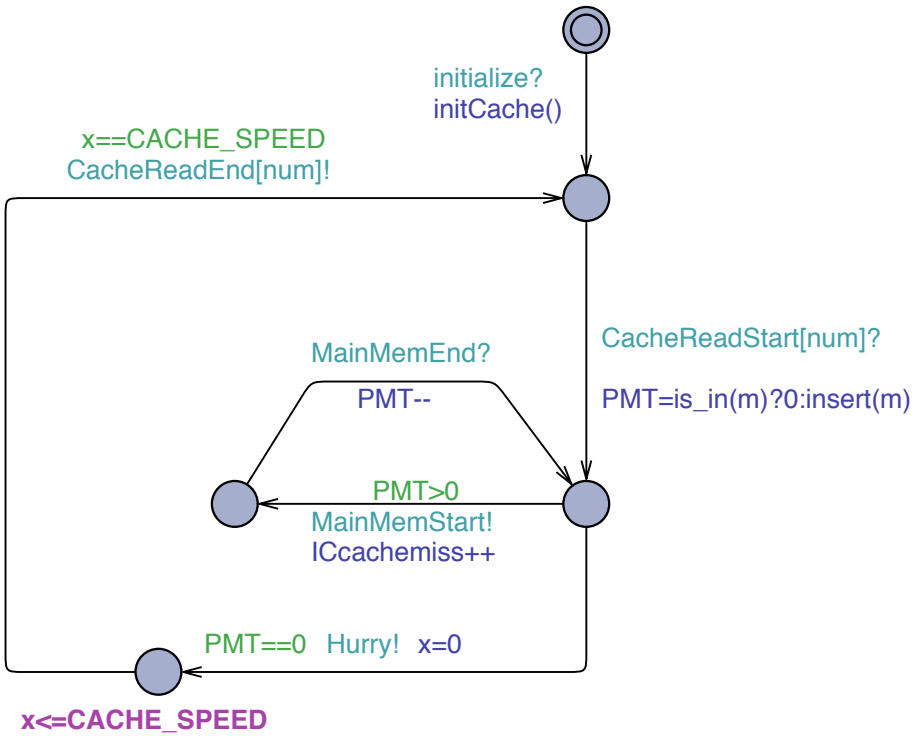


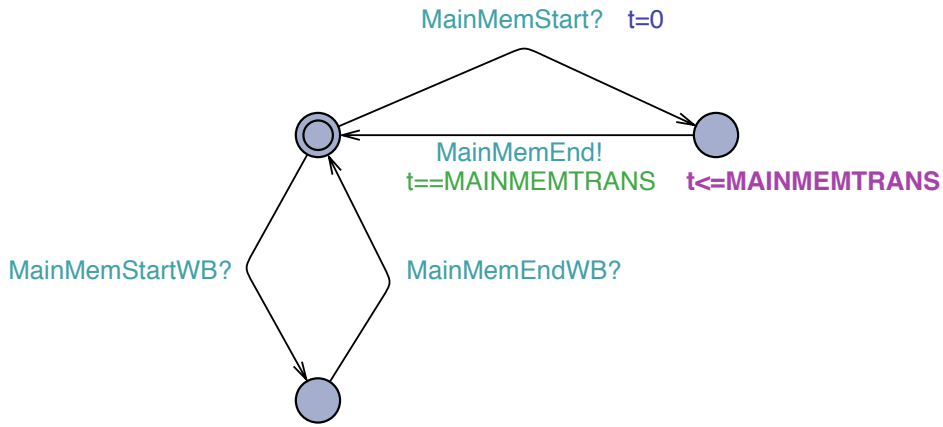


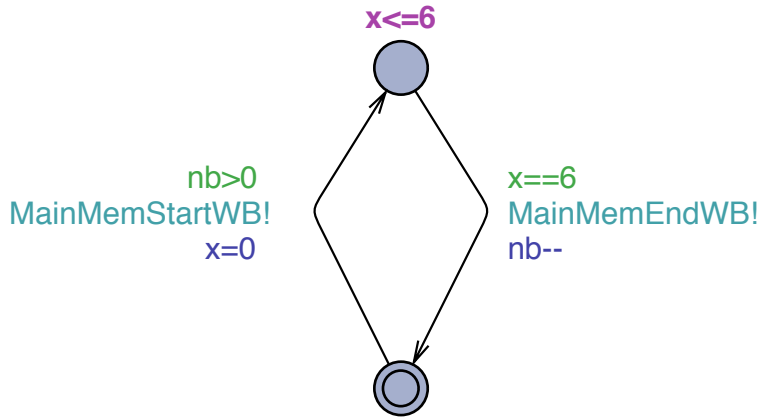
Pipeline Model – Memory Stage



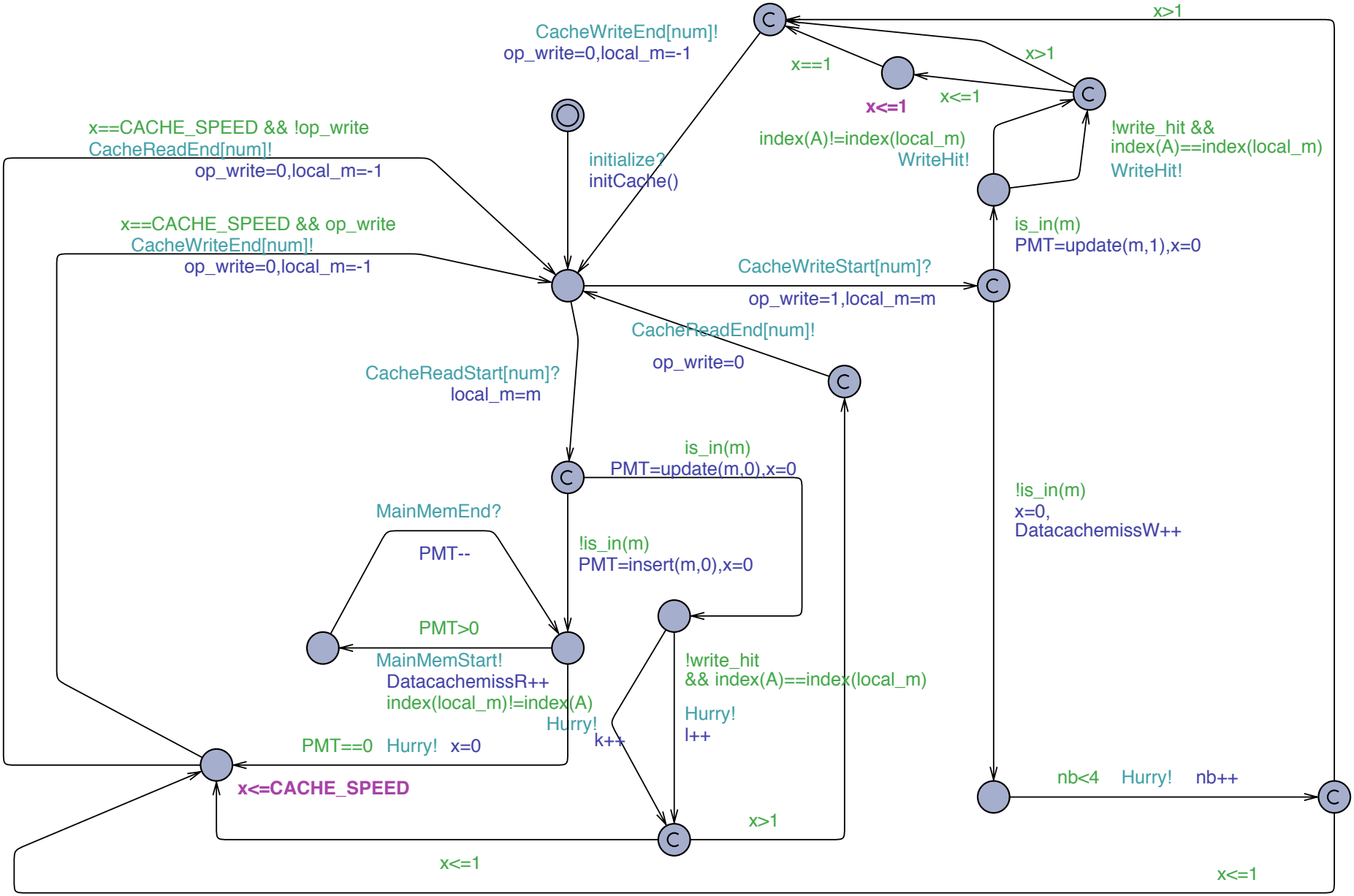
Instruction Cache & Main Memory



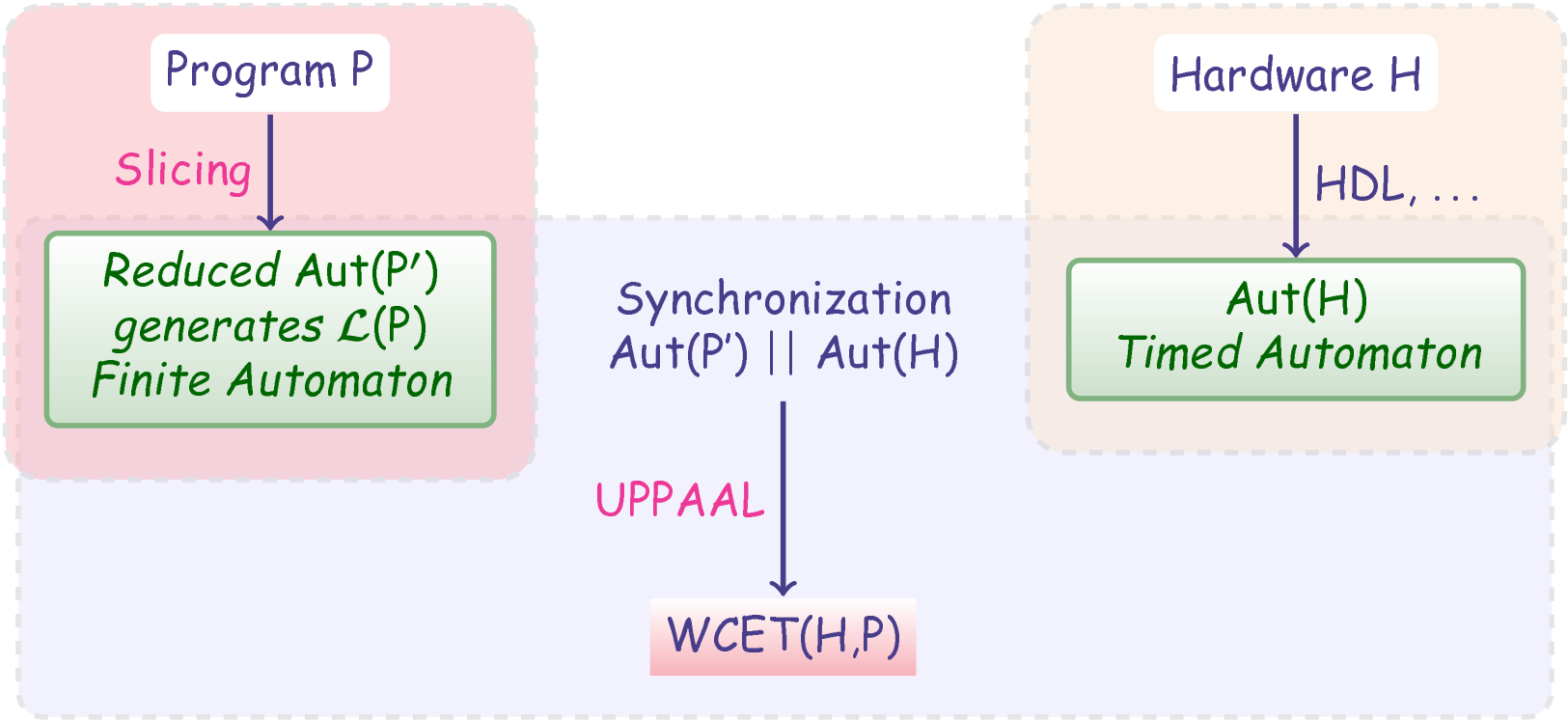




Data Cache


x>1
CacheWriteEnd[num]!
op_write=0,local_m=-1
x==1
x>1
x<=1
x<=1
x==CACHE_SPEED && !op_write
CacheReadEnd[num]!
op_write=0,local_m=-1
initialize?
initCache()
index(A)!=index(local_m)
WriteHit!
!write_hit &&
index(A)==index(local_m)
WriteHit!
x==CACHE_SPEED && op_write
CacheWriteEnd[num]!
op_write=0,local_m=-1
is_in(m)
PMT=update(m,1),x=0
CacheWriteStart[num]?
op_write=1,local_m=m
CacheReadEnd[num]!
op_write=0
CacheReadStart[num]?
local_m=m
is_in(m)
PMT=update(m,0),x=0
!is_in(m)
x=0,
DatacachemissW++
MainMemEnd?
PMT--
!is_in(m)
PMT=insert(m,0),x=0
PMT>0
MainMemStart!
DatacachemissR++
index(local_m)!=index(A)
Hurry!
k++
!write_hit
&& index(A)==index(local_m)
Hurry!
l++
PMT==0 Hurry! x=0
x<=CACHE_SPEED
nb<4 Hurry! nb++
x>1
x<=1
x<=1

Tool Chain



Slicing

[Tarjan, 1979] Thomas Lengauer and Robert Endre Tarjan.
A fast algorithm for finding dominators in a flowgraph.
ACM Trans. Program. Lang. Syst., 1(1):121–141, 1979.

Real-Time Model-Checking

[UPPAAL] K. G. Larsen, P. Pettersson, and W. Yi.
UPPAAL **in a Nutshell.**
Journal of Software Tools for Technology Transfer (STTT), 1(1-2):134–152, 1997.

Experiments & Results [Benchmarks, Mälardalen Univ.]

Program	loc	UPPAAL Time States Explored	Computed WCET (C)	Measured WCET (M)	$\frac{(C-M)}{M} \times 100$	Slice
Single-Path Programs						
fib-O0	74	1.74s/74181	8098	8064	**0.42%**	47/131
fib-O1	74	0.61s/22332	2597	2544	**2.0%**	18/72
fib-O2	74	0.3s/9710	1209	1164	**3.8%**	22/71
janne-complex-O0*	65	1.15s/38014	4264	4164	**2.4%**	78/173
janne-complex-O1*	65	0.48s/14600	1715	1680	**2.0%**	30/89
janne-complex-O2*	65	0.46s/13004	1557	1536	**1.3%**	32/78
fdct-O1	238	1.67s/60418	4245	4092	**3.7%**	100/363
fdct-O2	238	3.24s/55285	19231	18984	**1.3%**	166/3543
Single-Path Programs‡ with MUL/MLA/SMULL instructions (instructions durations depend on data)						
fdct-O0	238	2.41s/85007	[11242,11800]	11448	**3.0%**	253/831
matmult-O0*	162	5m9s/10531230	[502850,529250]	511584 528684	**0.1%**	158/314
matmult-O2*	162	43.78s/1780548	[122046,148299]	116844 140664	**5.4%**	75/288
jfdcint-O0	374	2.79s/100784	[12699,12699]	12588	**0.8%**	159/792
jfdcint-O1	374	1.02s/35518	[4897,4899]	4668	**7.0%**	25/325
jfdcint-O2	374	5.38s/175661	[16746,16938]	16380	**3.4%**	56/2512
Multiple-Path Programs						
bs-O0	174	42.6s/1421474	1068	1056	**1.1%**	75/151
bs-O1	174	28s/1214673	738	720	**2.5%**	28/82
bs-O2	174	15s/655870	628	600	**4.6%**	28/65
cnt-O0*	115	2.3s/76238	9028	8836	**2.1%**	99/235
cnt-O1*	115	1s/27279	4123	3996	**3.1%**	42/129
cnt-O2*	115	0.5s/11540	3065	2928	**4.6%**	39/263
insertsort-O0*	91	10m35s/24250737	3133	3108	**0.8%**	79/175
insertsort-O1*	91	7m2s/11455293	1533	1500	**2.2%**	40/115
insertsort-O2*	91	11.5s/387292	1371	1344	**2.0%**	43/108
ns-O0*	497	83.4s/3064315	30968	30732	**0.8%**	132/215
ns-O1*	497	11.3s/368719	11701	11568	**1.1%**	61/124
ns-O2*	497	29s/1030746	7343	7236	**1.4%**	566/863

Experiments & Results [Benchmarks, Mälardalen Univ.]

Program	loc	UPPAAL Time States Explored	Computed WCET (C)	Measured WCET (M)	$\frac{(C-M)}{M} \times 100$	Slice
Single-Path Programs						
fib-O0	74	1.74s/74181	8098	8064	**0.42%**	47/131
fib-O1	74	0.61s/22332	2597	2544	**2.0%**	18/72
fib-O2	74	0.3s/9710	1209	1164	**3.8%**	22/71
janne-complex-O0*	65	1.15s/38014	4264	4164	**2.4%**	78/173
janne-complex-O1*	65	0.48s/14600	1715	1680	**2.0%**	30/89
janne-complex-O2*	65	0.46s/13004	1557	1536	**1.3%**	32/78
fdct-O1	238	1.67s/60418	4245	4092	**3.7%**	100/363
fdct-O2	238	3.24s/55285	19231	18984	**1.3%**	166/3543
Single-Path Programs‡ with MUL/MLA/SMULL instructions (instructions durations depend on data)						
fdct-O0	238	2.41s/85007	[11242,11800]	11448	**3.0%**	253/831
matmult-O0*	162	5m9s/10531230	[502850,529250]	511584 528684	**0.1%**	158/314
matmult-O2*	162	43.78s/1780548	[122046,148299]	116844 140664	**5.4%**	75/288
jfdcint-O0	374	2.79s/100784	[12699,12699]	12588	**0.8%**	159/792
jfdcint-O1	374	1.02s/35518	[4897,4899]	4668	**7.0%**	25/325
jfdcint-O2	374	5.38s/175661	[16746,16938]	16380	**3.4%**	56/2512
Multiple-Path Programs						
bs-O0	174	42.6s/1421474	1068	1056	**1.1%**	75/151
bs-O1	174	28s/1214673	738	720	**2.5%**	28/82
bs-O2	174	15s/655870	628	600	**4.6%**	28/65
cnt-O0*	115	2.3s/76238	9028	8836	**2.1%**	99/235
cnt-O1*	115	1s/27279	4123	3996	**3.1%**	42/129
cnt-O2*	115	0.5s/11540	3065	2928	**4.6%**	39/263
insertsort-O0*	91	10m35s/24250737	3133	3108	**0.8%**	79/175
insertsort-O1*	91	7m2s/11455293	1533	1500	**2.2%**	40/115
insertsort-O2*	91	11.5s/387292	1371	1344	**2.0%**	43/108
ns-O0*	497	83.4s/3064315	30968	30732	**0.8%**	132/215
ns-O1*	497	11.3s/368719	11701	11568	**1.1%**	61/124
ns-O2*	497	29s/1030746	7343	7236	**1.4%**	566/863

Experiments & Results [Benchmarks, Mälardalen Univ.]

Program	loc	UPPAAL Time States Explored	Computed WCET (C)	Measured WCET (M)	$\frac{(C-M)}{M} \times 100$	Slice
Single-Path Programs						
fib-O0	74	1.74s/74181	8098	8064	**0.42%**	47/131
fib-O1	74	0.61s/22332	2597	2544	**2.0%**	18/72
fib-O2	74	0.3s/9710	1209	1164	**3.8%**	22/71
janne-complex-O0*	65	1.15s/38014	4264	4164	**2.4%**	78/173
janne-complex-O1*	65	0.48s/14600	1715	1680	**2.0%**	30/89
janne-complex-O2*	65	0.46s/13004	1557	1536	**1.3%**	32/78
fdct-O1	238	1.67s/60418	4245	4092	**3.7%**	100/363
fdct-O2	238	3.24s/55285	19231	18984	**1.3%**	166/3543
Single-Path Programs‡ with MUL/MLA/SMULL instructions (instructions durations depend on data)						
fdct-O0	238	2.41s/85007	[11242,11800]	11448	**3.0%**	253/831
matmult-O0*	162	5m9s/10531230	[502850,529250]	511584 528684	**0.1%**	158/314
matmult-O2*	162	43.78s/1780548	[122046,148299]	116844 140664	**5.4%**	75/288
jfdcint-O0	374	2.79s/100784	[12699,12699]	12588	**0.8%**	159/792
jfdcint-O1	374	1.02s/35518	[4897,4899]	4668	**7.0%**	25/325
jfdcint-O2	374	5.38s/175661	[16746,16938]	16380	**3.4%**	56/2512
Multiple-Path Programs						
bs-O0	174	42.6s/1421474	1068	1056	**1.1%**	75/151
bs-O1	174	28s/1214673	738	720	**2.5%**	28/82
bs-O2	174	15s/655870	628	600	**4.6%**	28/65
cnt-O0*	115	2.3s/76238	9028	8836	**2.1%**	99/235
cnt-O1*	115	1s/27279	4123	3996	**3.1%**	42/129
cnt-O2*	115	0.5s/11540	3065	2928	**4.6%**	39/263
insertsort-O0*	91	10m35s/24250737	3133	3108	**0.8%**	79/175
insertsort-O1*	91	7m2s/11455293	1533	1500	**2.2%**	40/115
insertsort-O2*	91	11.5s/387292	1371	1344	**2.0%**	43/108
ns-O0*	497	83.4s/3064315	30968	30732	**0.8%**	132/215
ns-O1*	497	11.3s/368719	11701	11568	**1.1%**	61/124
ns-O2*	497	29s/1030746	7343	7236	**1.4%**	566/863

Experiments & Results [Benchmarks, Mälardalen Univ.]

Program	loc	UPPAAL Time States Explored	Computed WCET (C)	Measured WCET (M)	$\frac{(C-M)}{M} \times 100$	Slice
Single-Path Programs						
fib-O0	74	1.74s/74181	8098	8064	**0.42%**	47/131
fib-O1	74	0.61s/22332	2597	2544	**2.0%**	18/72
fib-O2	74	0.3s/9710	1209	1164	**3.8%**	22/71
janne-complex-O0*	65	1.15s/38014	4264	4164	**2.4%**	78/173
janne-complex-O1*	65	0.48s/14600	1715	1680	**2.0%**	30/89
janne-complex-O2*	65	0.46s/13004	1557	1536	**1.3%**	32/78
fdct-O1	238	1.67s/60418	4245	4092	**3.7%**	100/363
fdct-O2	238	3.24s/55285	19231	18984	**1.3%**	166/3543
Single-Path Programs‡ with MUL/MLA/SMULL instructions (instructions durations depend on data)						
fdct-O0	238	2.41s/85007	[11242,11800]	11448	**3.0%**	253/831
matmult-O0*	162	5m9s/10531230	[502850,529250]	511584 528684	**0.1%**	158/314
matmult-O2*	162	43.78s/1780548	[122046,148299]	116844 140664	**5.4%**	75/288
jfdcint-O0	374	2.79s/100784	[12699,12699]	12588	**0.8%**	159/792
jfdcint-O1	374	1.02s/35518	[4897,4899]	4668	**7.0%**	25/325
jfdcint-O2	374	5.38s/175661	[16746,16938]	16380	**3.4%**	56/2512
Multiple-Path Programs						
bs-O0	174	42.6s/1421474	1068	1056	**1.1%**	75/151
bs-O1	174	28s/1214673	738	720	**2.5%**	28/82
bs-O2	174	15s/655870	628	600	**4.6%**	28/65
cnt-O0*	115	2.3s/76238	9028	8836	**2.1%**	99/235
cnt-O1*	115	1s/27279	4123	3996	**3.1%**	42/129
cnt-O2*	115	0.5s/11540	3065	2928	**4.6%**	39/263
insertsort-O0*	91	10m35s/24250737	3133	3108	**0.8%**	79/175
insertsort-O1*	91	7m2s/11455293	1533	1500	**2.2%**	40/115
insertsort-O2*	91	11.5s/387292	1371	1344	**2.0%**	43/108
ns-O0*	497	83.4s/3064315	30968	30732	**0.8%**	132/215
ns-O1*	497	11.3s/368719	11701	11568	**1.1%**	61/124
ns-O2*	497	29s/1030746	7343	7236	**1.4%**	566/863

Experiments & Results [Benchmarks, Mälardalen Univ.]

Program	loc	UPPAAL Time States Explored	Computed WCET (C)	Measured WCET (M)	$\frac{(C-M)}{M} \times 100$	Slice
Single-Path Programs						
fib-O0	74	1.74s/74181	8098	8064	**0.42%**	47/131
fib-O1	74	0.61s/22332	2597	2544	**2.0%**	18/72
fib-O2	74	0.3s/9710	1209	1164	**3.8%**	22/71
janne-complex-O0*	65	1.15s/38014	4264	4164	**2.4%**	78/173
janne-complex-O1*	65	0.48s/14600	1715	1680	**2.0%**	30/89
janne-complex-O2*	65	0.46s/13004	1557	1536	**1.3%**	32/78
fdct-O1	238	1.67s/60418	4245	4092	**3.7%**	100/363
fdct-O2	238	3.24s/55285	19231	18984	**1.3%**	166/3543
Single-Path Programs‡ with MUL/MLA/SMULL instructions (instructions durations depend on data)						
fdct-O0	238	2.41s/85007	[11242,11800]	11448	**3.0%**	253/831
matmult-O0*	162	5m9s/10531230	[502850,529250]	511584 528684	**0.1%**	158/314
matmult-O2*	162	43.78s/1780548	[122046,148299]	116844 140664	**5.4%**	75/288
jfdcint-O0	374	2.79s/100784	[12699,12699]	12588	**0.8%**	159/792
jfdcint-O1	374	1.02s/35518	[4897,4899]	4668	**7.0%**	25/325
jfdcint-O2	374	5.38s/175661	[16746,16938]	16380	**3.4%**	56/2512
Multiple-Path Programs						
bs-O0	174	42.6s/1421474	1068	1056	**1.1%**	75/151
bs-O1	174	28s/1214673	738	720	**2.5%**	28/82
bs-O2	174	15s/655870	628	600	**4.6%**	28/65
cnt-O0*	115	2.3s/76238	9028	8836	**2.1%**	99/235
cnt-O1*	115	1s/27279	4123	3996	**3.1%**	42/129
cnt-O2*	115	0.5s/11540	3065	2928	**4.6%**	39/263
insertsort-O0*	91	10m35s/24250737	3133	3108	**0.8%**	79/175
insertsort-O1*	91	7m2s/11455293	1533	1500	**2.2%**	40/115
insertsort-O2*	91	11.5s/387292	1371	1344	**2.0%**	43/108
ns-O0*	497	83.4s/3064315	30968	30732	**0.8%**	132/215
ns-O1*	497	11.3s/368719	11701	11568	**1.1%**	61/124
ns-O2*	497	29s/1030746	7343	7236	**1.4%**	566/863

Experiments & Results [Benchmarks, Mälardalen Univ.]

Program	loc	UPPAAL Time States Explored	Computed WCET (C)	Measured WCET (M)	$\frac{(C-M)}{M} \times 100$	Slice
Single-Path Programs						
fib-O0	74	1.74s/74181	8098	8064	**0.42%**	47/131
fib-O1	74	0.61s/22332	2597	2544	**2.0%**	18/72
fib-O2	74	0.3s/9710	1209	1164	**3.8%**	22/71
janne-complex-O0*	65	1.15s/38014	4264	4164	**2.4%**	78/173
janne-complex-O1*	65	0.48s/14600	1715	1680	**2.0%**	30/89
janne-complex-O2*	65	0.46s/13004	1557	1536	**1.3%**	32/78
fdct-O1	238	1.67s/60418	4245	4092	**3.7%**	100/363
fdct-O2	238	3.24s/55285	19231	18984	**1.3%**	166/3543
Single-Path Programs‡ with MUL/MLA/SMULL instructions (instructions durations depend on data)						
fdct-O0	238	2.41s/85007	[11242,11800]	11448	**3.0%**	253/831
matmult-O0*	162	5m9s/10531230	[502850,529250]	511584 528684	**0.1%**	158/314
matmult-O2*	162	43.78s/1780548	[122046,148299]	116844 140664	**5.4%**	75/288
jfdcint-O0	374	2.79s/100784	[12699,12699]	12588	**0.8%**	159/792
jfdcint-O1	374	1.02s/35518	[4897,4899]	4668	**7.0%**	25/325
jfdcint-O2	374	5.38s/175661	[16746,16938]	16380	**3.4%**	56/2512
Multiple-Path Programs						
bs-O0	174	42.6s/1421474	1068	1056	**1.1%**	75/151
bs-O1	174	28s/1214673	738	720	**2.5%**	28/82
bs-O2	174	15s/655870	628	600	**4.6%**	28/65
cnt-O0*	115	2.3s/76238	9028	8836	**2.1%**	99/235
cnt-O1*	115	1s/27279	4123	3996	**3.1%**	42/129
cnt-O2*	115	0.5s/11540	3065	2928	**4.6%**	39/263
insertsort-O0*	91	10m35s/24250737	3133	3108	**0.8%**	79/175
insertsort-O1*	91	7m2s/11455293	1533	1500	**2.2%**	40/115
insertsort-O2*	91	11.5s/387292	1371	1344	**2.0%**	43/108
ns-O0*	497	83.4s/3064315	30968	30732	**0.8%**	132/215
ns-O1*	497	11.3s/368719	11701	11568	**1.1%**	61/124
ns-O2*	497	29s/1030746	7343	7236	**1.4%**	566/863

Summary

Fully automatic computation of WCET

- Computation of CFG of binary programs + reduced program
  Program slicing
- Formal models of hardware (pipeline and caches)
  Identification of hardware features
- Computation of WCET as a reachability property
  Real-time model-checking with UPPAAL

Experiments to evaluate tightness of results

- method to measure execution-times on ARM920T
- evaluation on benchmarks from Mälardalen University, Sweden
- over-approximation is less than 5%

Advantages of our method

- Modular
- Fully automatic
- Can easily accommodate new features

Summary

Fully automatic computation of WCET

- Computation of CFG of binary programs + reduced program
  Program slicing
- Formal models of hardware (pipeline and caches)
  Identification of hardware features
- Computation of WCET as a reachability property
  Real-time model-checking with UPPAAL

Experiments to evaluate tightness of results

- method to measure execution-times on ARM920T
- evaluation on benchmarks from Mälardalen University, Sweden
- over-approximation is less than 5%

Advantages of our method

- Modular
- Fully automatic
- Can easily accommodate new features

Summary

Fully automatic computation of WCET

- Computation of CFG of binary programs + reduced program
  Program slicing
- Formal models of hardware (pipeline and caches)
  Identification of hardware features
- Computation of WCET as a reachability property
  Real-time model-checking with UPPAAL

Experiments to evaluate tightness of results

- method to measure execution-times on ARM920T
- evaluation on benchmarks from Mälardalen University, Sweden
- over-approximation is less than 5%

Advantages of our method

- Modular
- Fully automatic
- Can easily accommodate new features

Ongoing and Future Work

New Features

- Processor speed changes
- For OS programs, model for interruptions' arrivals

New Architectures

- models of PowerPC (multi-core)
- refine cache models

Enhanced Analysis (model-checking)

- Compute a witness trace that gives the WCET
- Refinement CEGAR
- Design a customized real-time model-checker
  taking advantage of particular features of the WCET problem
- reduce the reduced program
  reduce number of paths to explore

Tool Release forthcoming (`http://www.irccyn.fr/franck/wcet`)

- Command line compiler from binary programs to UPPAAL
- Qt Interface

Ongoing and Future Work

New Features

- Processor speed changes
- For OS programs, model for interruptions' arrivals

New Architectures

- models of PowerPC (multi-core)
- refine cache models

Enhanced Analysis (model-checking)

- Compute a witness trace that gives the WCET
- Refinement CEGAR
- Design a customized real-time model-checker
  taking advantage of particular features of the WCET problem
- reduce the reduced program
  reduce number of paths to explore

Tool Release forthcoming (`http://www.irccyn.fr/franck/wcet`)

- Command line compiler from binary programs to UPPAAL
- Qt Interface

Ongoing and Future Work

New Features

- Processor speed changes
- For OS programs, model for interruptions' arrivals

New Architectures

- models of PowerPC (multi-core)
- refine cache models

Enhanced Analysis (model-checking)

- Compute a witness trace that gives the WCET
- Refinement CEGAR
- Design a customized real-time model-checker
  taking advantage of particular features of the WCET problem
- reduce the reduced program
  reduce number of paths to explore

Tool Release forthcoming (`http://www.irccyn.fr/franck/wcet`)

- Command line compiler from binary programs to UPPAAL
- Qt Interface

Ongoing and Future Work

New Features

- Processor speed changes
- For OS programs, model for interruptions' arrivals

New Architectures

- models of PowerPC (multi-core)
- refine cache models

Enhanced Analysis (model-checking)

- Compute a witness trace that gives the WCET
- Refinement CEGAR
- Design a customized real-time model-checker
  taking advantage of particular features of the WCET problem
- reduce the reduced program
  reduce number of paths to explore

Tool Release forthcoming (`http://www.irccyn.fr/franck/wcet`)

- Command line compiler from binary programs to UPPAAL
- Qt Interface

Is Slicing Critical ?

METAMOC [Master's Thesis 2009, WCET'2010]

- Mälardalen University Sweden and Aalborg Univ. Denmark
- Timed automata (hardware model)
- Annotate with loop bounds
- Value analysis phase
- Loop unfolding
- Compute WCET using UPPAAL

Results (from **http://metamoc.dk/**)

- Programs compiled with O2 option
- Simple cache formal models

Is Slicing Critical ?

WCET2010 - LRU

	ATMEL	ARM9	ARM9	ARM9	ARM9	ARM9	ARM9
Optimization	O2	O2	O2	O2	O2	O2	O2
Data-cache	-	miss_writeback	miss_writeback	miss_writeback	Concrete, 64 lines/set, LRU	Concrete, 128 lines/set, LRU	Concrete, LRU
Instr.-cache	-	miss	Concrete, 128 lines/set, LRU	Concrete, LRU	Concrete, LRU	Concrete, LRU	Concrete, LRU
Value-analysis	No	No	No	No	Yes	Yes	Yes
adpcm	**OOM**	**19829472**		**OOM/error**			**OOM/error**
	1:44:19	20:12.95		1:46.67			4:44.83
bs	**146**	**3666**		**964**			**640**
	0:03.66	0:01.09		0:01.06			0:01.27
bsort100		**7997421**		**3425868**	**408715**	**OOM**	**OOM/error**
		0:29.54		0:57.66	2:06.05	2:02.95	1:50.13
cnt	**29572**	**188466**		**52481**			**6137**
	0:02.46	0:02.06		0:02.62			0:04.00
compress	**58352**	**158697**		**69488**		**Model invalid, manual mod.**	**60351**
	6:29.88	0:06.52		0:09.11		0:04.56+0:22.65	0:27.21
crc	**170959**	**2043591**		**328310**			**310722**
	0:36.71	0:13.42		0:26.32			0:39.83
edn	**345499**	**2093937**		**687463**			**431334**
	0:29.42	0:12.81		0:20.56			4:20.71
expint	**7583931**	**652215**		**31483**			**30577**
	4:56.17	0:04.33		0:08.75			0:12.72
fac	**906**	**11553**		**1266**			**683**
	0:01.43	0:01.05		0:01.02			0:01.35
fdct	**4806**	**344234**		**190692**			**168827**
	0:02.43	0:07.10		0:08.26			41:22.70
fibcall	**438**	**13434**		**632**			**599**
	0:01.04	0:01.09		0:01.08			0:01.52
fir	**782494**	**131508**		**21364**			**18763**
	0:25.90	0:02.02		0:02.73			0:04.53
insertsort	**2872**	**75078**		**43888**			**37814**
	0:01.26	0:01.20		0:01.31			0:02.44
janne_complex	**OOM**	**57787**		**1883**			**1883**
	2:11:27	0:01.17		0:01.33			0:01.92
jfdctint	**54383**	**294957**		**133059**			**111510**
	0:05.80	0:05.39		0:06.41			16:46.68
matmult	**525383**	**5836449**		**2636715**		**712869**	**OOM/error**
	0:17.02	0:22.16		0:40.72		1:28.47	1:36.94
ndes	**OOM**	**2575484**	**1084016**	**OOM/error**			**OOM/error**
	26:47.04	2:55.38	7:33.04	8:27.09			4:25.09
ns	**13116**	**279579**		**29060**			**10813**
	0:01.79	0:02.20		0:03.43			0:07.09
nsichneu		**1020957**		**622904**			**295997**
		3:04.11		3:25.56			3:37.70
prime	**170806**	**6477669**		**470372**			**469060**
	0:12.52	1:02.36		1:56.36			2:41.87
ud	**58217**	**1160049**		**175359**			**165814**
	0:11.81	0:07.76		0:14.64			0:23.07
	3 errors / 19 benchmarks	0 errors / 21 benchmar	1 errors / 21 benchmarks	errors / 21 benchmarks	2 errors / 21 benchmarks	3 errors / 21 benchmarks	4 errors / 21 benchmarks
	Model checking fails: 3		Model checking fails: 1 Manual modification: 1	Model checking fails: 2 Manual modification: 0	Value analysis fails: 0 Model checking fails: 2 Manual modification: 3	Value analysis fails: 0 Model checking fails: 3 Manual modification: 2	Value analysis fails: 0 Model checking fails: 4 Manual modification: 1

Program Slicing, M. Weiser [Weiser, 1984]

I = set of instructions in P

Slicing

- slice criterion: subset $I' \subseteq I$ and variables $\mathcal{V}(i)$ for each $i \in I'$
  `4:str r0,[sp,#4]` and variable `sp`
- Slice of P = sub-program P′ of P satisfying (1) and (2)
  - given input $d \in \mathcal{D}$,

    run of P on d $\quad \rho = (i_0, v_0)\ (i_1, v_1)\ \cdots\ (i_k, v_k)\ \cdots\ (i_n, v_n)$

    run of P′ on d $\quad \rho' = (i'_0, v'_0)\ (i'_1, v'_1)\ \cdots\ (i'_l, v'_l)\ \cdots\ (i'_m, v'_m)$

  - projection: for a pair (i, v), $\mathrm{proj}(i, v) = \begin{cases} \varepsilon \text{ if } i \notin I' \\ (i, \mathrm{proj}_{\mathcal{V}(i)}(v)) \text{ otherwise} \end{cases}$
  - for sequences of pairs: $\mathrm{proj}^*(\varepsilon) = \varepsilon$ and $\mathrm{proj}^*(w.a) = \mathrm{proj}^*(w).\mathrm{proj}(a)$

  1. on input $d \in \mathcal{D}$, if P terminates then P′ terminates
  2. on input $d \in \mathcal{D}$, $\mathrm{proj}^*(\rho) = \mathrm{proj}^*(\rho')$
- a sub-program P′ can be effectively computed (no optimal one)
  compute data dependences and control dependences

Program Slicing, M. Weiser [Weiser, 1984]

I = set of instructions in P

Slicing

- slice criterion: subset $I' \subseteq I$ and variables $\mathcal{V}(i)$ for each $i \in I'$
  `4:str r0,[sp,#4]` and variable `sp`
- Slice of P = sub-program P′ of P satisfying (1) and (2)
  - given input $d \in \mathcal{D}$,

$$\begin{array}{rlllll} \text{run of P on d} & \varrho = (i_0, v_0) & (i_1, v_1) & \cdots & (i_k, v_k) & \cdots & (i_n, v_n) \\ \text{run of P′ on d} & \varrho' = (i'_0, v'_0) & (i'_1, v'_1) & \cdots & (i'_l, v'_l) & \cdots & (i'_m, v'_m) \end{array}$$

  - projection: for a pair (i, v), $\text{proj}(i, v) = \begin{cases} \varepsilon \text{ if } i \notin I' \\ (i, \text{proj}_{\mathcal{V}(i)}(v)) \text{ otherwise} \end{cases}$
  - for sequences of pairs: $\text{proj}^*(\varepsilon) = \varepsilon$ and $\text{proj}^*(w.a) = \text{proj}^*(w).\text{proj}(a)$

  1. on input $d \in \mathcal{D}$, if P terminates then P′ terminates
  2. on input $d \in \mathcal{D}$, $\text{proj}^*(\varrho) = \text{proj}^*(\varrho')$

- a sub-program P′ can be effectively computed (no optimal one)
  compute data dependences and control dependences

Program Slicing, M. Weiser [Weiser, 1984]

I = set of instructions in P

Slicing

- slice criterion: subset $I' \subseteq I$ and variables $\mathcal{V}(i)$ for each $i \in I'$
  `4:str r0,[sp,#4]` and variable `sp`
- Slice of P = sub-program P′ of P satisfying (1) and (2)
  - given input $d \in \mathcal{D}$,

$$\begin{array}{lccccccc} \text{run of P on d} & \varrho = (i_0,v_0) & (i_1,v_1) & \cdots & (i_k,v_k) & \cdots & (i_n,v_n) \\ \text{run of P′ on d} & \varrho' = (i'_0,v'_0) & (i'_1,v'_1) & \cdots & (i'_l,v'_l) & \cdots & (i'_m,v'_m) \end{array}$$

  - projection: for a pair (i,v), $\text{proj}(i,v) = \begin{cases} \varepsilon \text{ if } i \notin I' \\ (i, \text{proj}_{\mathcal{V}(i)}(v)) \text{ otherwise} \end{cases}$
  - for sequences of pairs: $\text{proj}^*(\varepsilon) = \varepsilon$ and $\text{proj}^*(w.a) = \text{proj}^*(w).\text{proj}(a)$

  1. on input $d \in \mathcal{D}$, if P terminates then P′ terminates
  2. on input $d \in \mathcal{D}$, $\text{proj}^*(\varrho) = \text{proj}^*(\varrho')$

- a sub-program P′ can be effectively computed (no optimal one)
  compute data dependences and control dependences

Computing a WCET-equivalent Slice

Assume CFG of P is known

Slice criterion C

- each memory transfer instruction is in the criterion C
  `32:ldr r2,[r1,#4]` (32,r1)
- each conditional branch instruction is in C
  `36:beq 34`

What's in the Slice ?

1. initially slice $S = C$ (memory transfers and conditional branching)
2. add to S instructions and variables that define the values of vars in S
   e.g., `28:add r1,r3,#1`
3. add to S instructions and variables that influence the control flow
   e.g., "hidden" loop counters, variables used in comparisons
4. repeat from (2) until fixpoint is reached

Key Result [Weiser 1984]
A slice can be automatically computed.

Computing a WCET-equivalent Slice

Assume CFG of P is known

Slice criterion C

- each memory transfer instruction is in the criterion C
  `32:ldr r2,[r1,#4]` (32,r1)
- each conditional branch instruction is in C
  `36:beq 34`

What's in the Slice ?

1. initially slice S = C (memory transfers and conditional branching)
2. add to S instructions and variables that define the values of vars in S
   e.g., `28:add r1,r3,#1`
3. add to S instructions and variables that influence the control flow
   e.g., "hidden" loop counters, variables used in comparisons
4. repeat from (2) until fixpoint is reached

Key Result [Weiser 1984]
A slice can be automatically computed.

Computing a WCET-equivalent Slice

Assume CFG of P is known

Slice criterion C

- each memory transfer instruction is in the criterion C
  `32:ldr r2,[r1,#4]` (32,r1)
- each conditional branch instruction is in C
  `36:beq 34`

What's in the Slice ?

1. initially slice S = C (memory transfers and conditional branching)
2. add to S instructions and variables that define the values of vars in S
   e.g., `28:add r1,r3,#1`
3. add to S instructions and variables that influence the control flow
   e.g., "hidden" loop counters, variables used in comparisons
4. repeat from (2) until fixpoint is reached

Key Result [Weiser 1984]
A slice can be automatically computed.

Measuring Execution-Time on the ARM920T

```
#define timerToCPUClockRatio 12

main ()
{
  int result;
  unsigned int start;
  unsigned int stop;

  start = timerGetValue(1);
  result = fib(300);
  stop = timerGetValue(1);
  printf("fib(300):␣%d,␣time=%lu\n", result,
                (stop-start)*timerToCPUClockRatio);
  while (1);
}
```

- Embedded hardware timer: 1/12th of processor clock frequency
- measurement error is ± 24 cycles
- a program executing in ≥ 1200 cycles may be accurately measured less than 1% of measurement error

Compiled Program

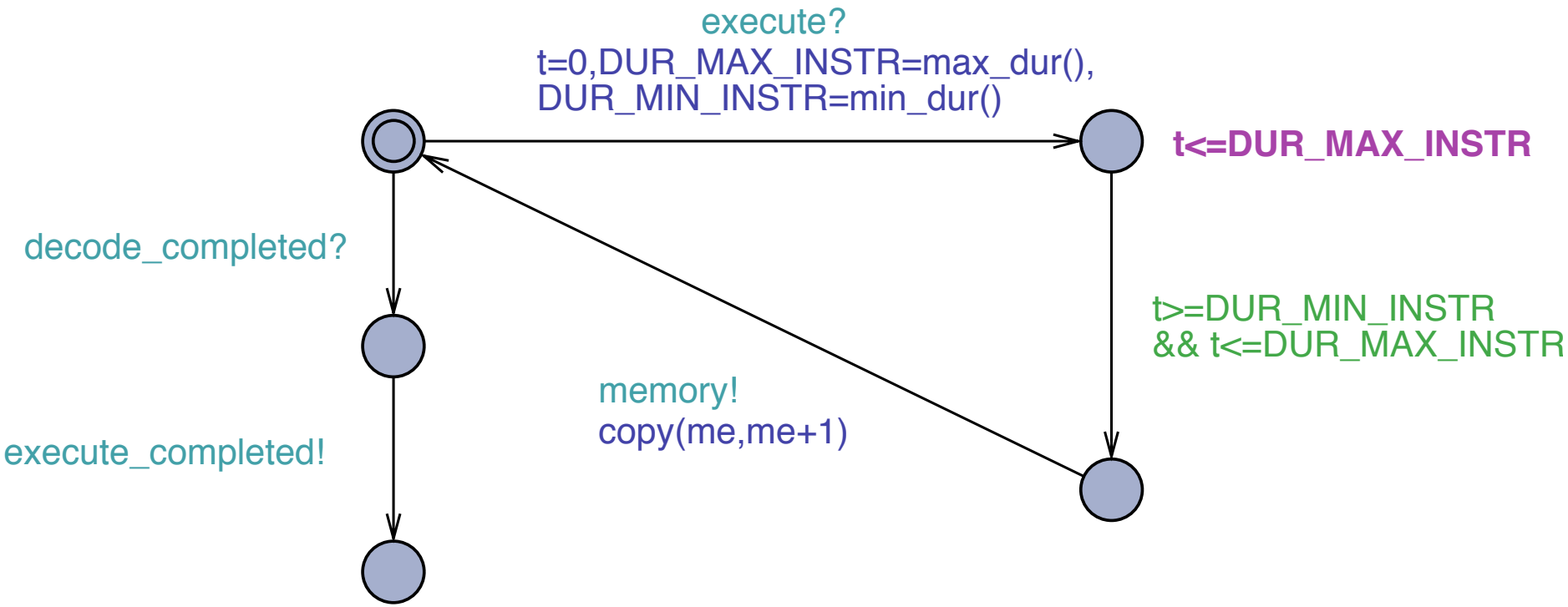
```
00003214 <fib>:
 0003214: e24dd020  sub sp, sp, #32
 0003218: e58d0004  str r0, [sp, #4]
 000321c: e3a03001  mov r3, #1
 .....
 0003288: e12fff1e  bx lr

0000328c <printbin>:
 000328c: e52de004  bx lr

0000332c <timerGetRegisterAddress>:
 000332c: e24dd008  sub sp, sp, #8
 0003330: e58d0004  str r0, [sp, #4]
 0003334: e58d1000  str r1, [sp]
 0003338: e59d3004  ldr r3, [sp, #4]
 000333c: e3530001  cmp r3, #1
 0003340: 1a000003  bne 0003354 <timerGetRegisterAddress+0x28>
 0003344: e59d3000  ldr r3, [sp]
 0003348: e2833602  add r3, r3, #2097152 ; 0x200000
 000334c: e2833a02  add r3, r3, #8192 ; 0x2000
 0003350: ea000002  b  0003360 <timerGetRegisterAddress+0x34>
 0003354: e59d3000  ldr r3, [sp]
 0003358: e2833602  add r3, r3, #2097152 ; 0x200000
 000335c: e2833a03  add r3, r3, #12288 ; 0x3000
 0003360: e1a00003  mov r0, r3
 0003364: e28dd008  add sp, sp, #8
 0003368: e12fff1e  bx lr

0000336c <timerInit>:
 000336c: e52de004 bx  lr

000033a8 <timerSetPrescaler>:
 00033a8: e52de004 bx  lr
```

```
000033e4 <timerGetValue>:
 00033e4: e52de004  push {lr}
 00033e8: e24dd014  sub sp, sp, #20
 00033ec: e58d0004  str r0, [sp, #4]
 00033f0: e59d0004  ldr r0, [sp, #4]
 00033f4: e3a01010  mov r1, #16
 00033f8: ebffffcb  bl 000332c <timerGetRegisterAddress>
 00033fc: e1a03000  mov r3, r0
 0003400: e58d300c  str r3, [sp, #12]
 0003404: e59d300c  ldr r3, [sp, #12]
 0003408: e5933000  ldr r3, [r3]
 000340c: e58d3008  str r3, [sp, #8]
 0003410: e59d3008  ldr r3, [sp, #8]
 0003414: e1a00003  mov r0, r3
 0003418: e28dd014  add sp, sp, #20
 000341c: e49df004  pop {pc}

00004ce8 <main>:
 0004ce8: e52de004  push {lr}
 0004cec: e24dd014  sub sp, sp, #20
 0004cf0: e3a00001  mov r0, #1
 0004cf4: e3a01002  mov r1, #2
 0004cf8: ebfff99b  bl 000336c <timerInit>
 0004cfc: e3a00001  mov r0, #1
 0004d00: e3a01000  mov r1, #0
 0004d04: ebfff9a7  bl 00033a8 <timerSetPrescaler>
 0004d08: ebfffe64  mov r0, r0
 ...
 0004d44: ebfff9a6  bl 00033e4 <timerGetValue>
 0004d48: e1a03000  mov r3, r0
 0004d4c: e58d3004  str r3, [sp, #4]
 0004d50: e3a00f4b  mov r0, #300
 0004d54: ebfff92e  bl 0003214 <fib>
 0004d58: e1a03000  mov r3, r0
 0004d5c: e58d3000  str r3, [sp]
 0004d60: e3a00001  mov r0, #1
 0004d64: ebfff99e  bl 00033e4 <timerGetValue>
 0004d68: e1a03000  mov r3, #3
 ...
```

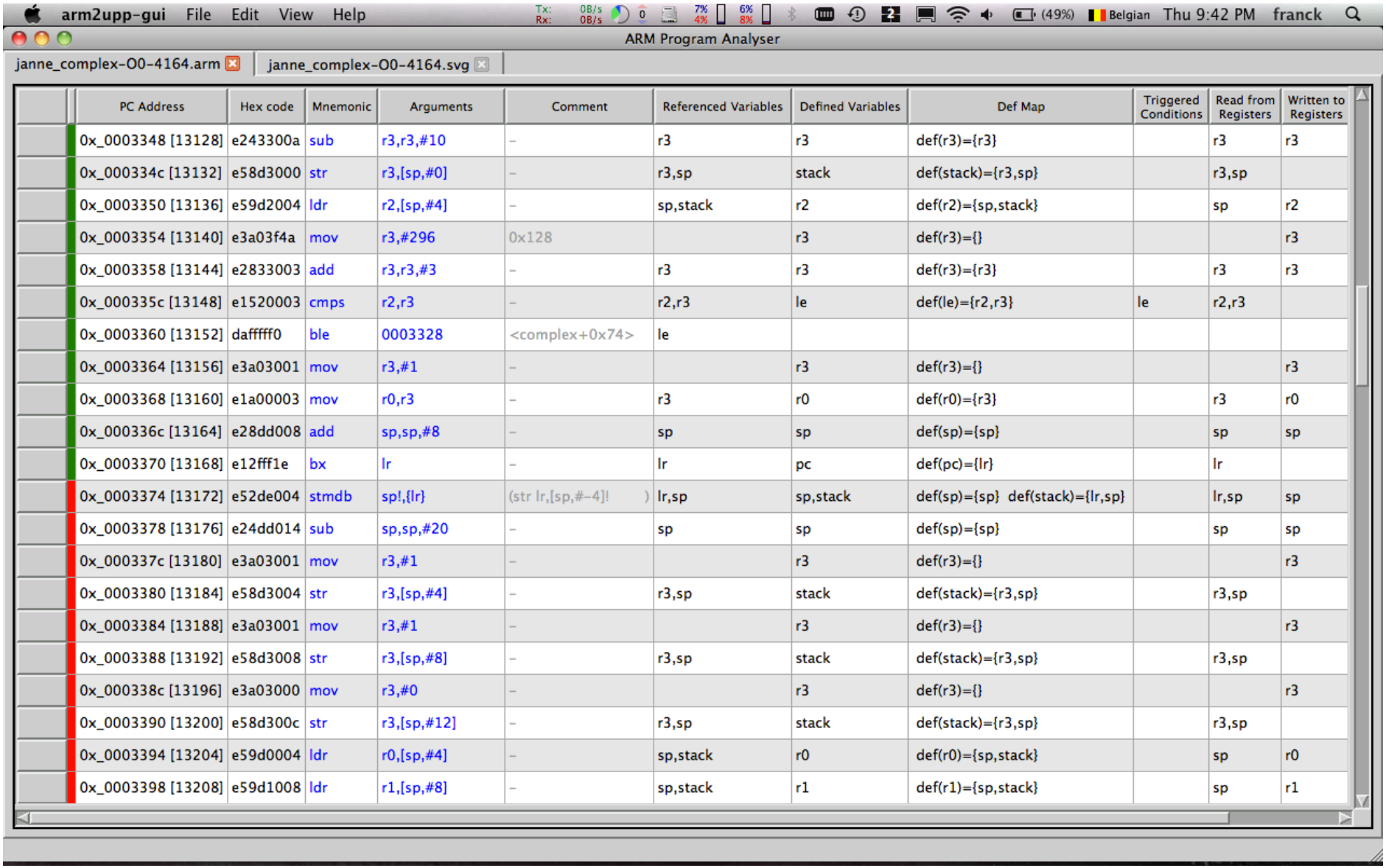
Compiled Program

```
...
0004d44: ebfff9a6  bl 00033e4 <timerGetValue>
0004d48: e1a03000  mov r3, r0
0004d4c: e58d3004  str r3, [sp, #4]
0004d50: e3a00f4b  mov r0, #300
0004d54: ebfff92e  bl 0003214 <fib>
0004d58: e1a03000  mov r3, r0
0004d5c: e58d3000  str r3, [sp]
0004d60: e3a00001  mov r0, #1
0004d64: ebfff99e  bl 00033e4 <timerGetValue>
0004d68: e1a03000  mov r3, #3
...
```

Interval in Execute Stage



Qt GUI



PC Address	Hex code	Mnemonic	Arguments	Comment	Referenced Variables	Defined Variables	Def Map	Triggered Conditions	Read from Registers	Written to Registers
0x_0003348 [13128]	e243300a	sub	r3,r3,#10	-	r3	r3	def(r3)={r3}		r3	r3
0x_000334c [13132]	e58d3000	str	r3,[sp,#0]	-	r3,sp	stack	def(stack)={r3,sp}		r3,sp	
0x_0003350 [13136]	e59d2004	ldr	r2,[sp,#4]	-	sp,stack	r2	def(r2)={sp,stack}		sp	r2
0x_0003354 [13140]	e3a03f4a	mov	r3,#296	0x128		r3	def(r3)={}			r3
0x_0003358 [13144]	e2833003	add	r3,r3,#3	-	r3	r3	def(r3)={r3}		r3	r3
0x_000335c [13148]	e1520003	cmps	r2,r3	-	r2,r3	le	def(le)={r2,r3}	le	r2,r3	
0x_0003360 [13152]	daffffff0	ble	0003328	<complex+0x74>	le					
0x_0003364 [13156]	e3a03001	mov	r3,#1	-		r3	def(r3)={}			r3
0x_0003368 [13160]	e1a00003	mov	r0,r3	-	r3	r0	def(r0)={r3}		r3	r0
0x_000336c [13164]	e28dd008	add	sp,sp,#8	-	sp	sp	def(sp)={sp}		sp	sp
0x_0003370 [13168]	e12fff1e	bx	lr	-	lr	pc	def(pc)={lr}		lr	
0x_0003374 [13172]	e52de004	stmdb	sp!,{lr}	(str lr,[sp,#-4]!)	lr,sp	sp,stack	def(sp)={sp} def(stack)={lr,sp}		lr,sp	sp
0x_0003378 [13176]	e24dd014	sub	sp,sp,#20	-	sp	sp	def(sp)={sp}		sp	sp
0x_000337c [13180]	e3a03001	mov	r3,#1	-		r3	def(r3)={}			r3
0x_0003380 [13184]	e58d3004	str	r3,[sp,#4]	-	r3,sp	stack	def(stack)={r3,sp}		r3,sp	
0x_0003384 [13188]	e3a03001	mov	r3,#1	-		r3	def(r3)={}			r3
0x_0003388 [13192]	e58d3008	str	r3,[sp,#8]	-	r3,sp	stack	def(stack)={r3,sp}		r3,sp	
0x_000338c [13196]	e3a03000	mov	r3,#0	-		r3	def(r3)={}			r3
0x_0003390 [13200]	e58d300c	str	r3,[sp,#12]	-	r3,sp	stack	def(stack)={r3,sp}		r3,sp	
0x_0003394 [13204]	e59d0004	ldr	r0,[sp,#4]	-	sp,stack	r0	def(r0)={sp,stack}		sp	r0
0x_0003398 [13208]	e59d1008	ldr	r1,[sp,#8]	-	sp,stack	r1	def(r1)={sp,stack}		sp	r1

References I

[1] Armadeus systems.

[2] ARM9TDMI Technical Reference Manual.
ARM Limited, 2000.

[3] Application Binary Interface for the ARM Architecture.
ARM Limited, 2009.

[4] AbsInt Angewandte Informatik.
aiT Worst-Case Execution Time Analyzers.
http://www.absint.com/ait/.

[5] R. Alur and D. Dill.
A theory of timed automata.
Theoretical Computer Science, 126(2):183–235, 1994.

[6] ARM Limited.
Application Note 93 – Benchmarking with ARMulator.
http://infocenter.arm.com/help/topic/com.arm.doc.dai0093a/DAI0093A_benchmarking_appsnote.pdf.

[7] Clément Ballabriga, Hugues Cassé, Christine Rochange, and Pascal Sainrat.
Otawa: An open toolbox for adaptive wcet analysis.
In Sang Lyul Min, Robert G. Pettit IV, Peter P. Puschner, and Theo Ungerer, editors, Software Technologies for Embedded and Ubiquitous Systems (SEUS) - 8th IFIP WG 10.2 International Workshop, SEUS 2010, Waidhofen/Ybbs, Austria, October 13-15, 2010. Proceedings, volume 6399 of LNCS, pages 35–46. Springer, 2010.

References II

[8] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks.
Uppaal 4.0.
In QEST, pages 125–126. IEEE Computer Society, 2006.

[9] G. Bernat, A. Colin, and S. M. Petters.
pWCET a Toolset for automatic Worst-Case Execution Time Analysis of Real-Time Embedded Programs.
In Proceedings of the 3rd Int. Workshop on WCET Analysis, Workshop of the Euromicro Conference on Real-Time Systems, Porto, Portugal, 2003.

[10] Franck Cassez.
Timed Games for Computing WCET for Pipelined Processors with Caches.
In 11th Int. Conf. on Application of Concurrency to System Design (ACSD'11). IEEE Comp. Soc., June 2011.
forthcoming.

[11] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith.
Counterexample-guided abstraction refinement for symbolic model checking.
J. ACM, 50(5):752–794, 2003.

[12] Codesourcery.
Web site.
http://www.codesourcery.com/.

[13] Keith D. Cooper, Timothy J. Harvey, and Ken Kennedy.
A Simple, Fast Dominance Algorithm.
Software – Practice and Experience, 4:1–10, 2001.

References III

[WCET'2010] Andreas E. Dalsgaard, Mads Chr. Olesen, Martin Toft, René Rydhof Hansen, and Kim Guldstrand Larsen.
Metamoc: Modular execution time analysis using model checking.
In Björn Lisper, editor, WCET, volume 15 of OASICS, pages 113–123. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2010.

[Master's Thesis 2009] Andreas Engelbredt Dalsgaard, Mads Christian Olesen, and Martin Toft.
Modular execution time analysis using model checking.
Master's thesis, Department of Computer Science, Aalborg University, Denmark, 2009.

[14] Jakob Engblom, Andreas Ermedahl, Mikael Nolin, Jan Gustafsson, and Hans Hansson.
Worst-case execution-time analysis for embedded real-time systems.
Journal on Software Tools for Technology Transfer (STTT), 4(4):437–455, October 2003.

[15] Christian Ferdinand, Reinhold Heckmann, and Reinhard Wilhelm.
Analyzing the worst-case execution time by abstract interpretation of executable code.
In Manfred Broy, Ingolf H. Krüger, and Michael Meisinger, editors, ASWSD, volume 4147 of LNCS, pages 1–14. Springer, 2004.

[16] Loukas Georgiadis, Robert Endre Tarjan, and Renato Fonseca F. Werneck.
Finding dominators in practice.
J. Graph Algorithms Appl., 10(1):69–94, 2006.

[17] Jan Gustafsson, Adam Betts, Andreas Ermedahl, and Björn Lisper.
The Mälardalen WCET benchmarks – past, present and future.
pages 137–147, Brussels, Belgium, July 2010. OCG.

References IV

[18] Niklas Holsti, Jan Gustafsson, Guillem Bernat, Clément Ballabriga, Armelle Bonenfant, Roman Bourgade, Hugues Cassé, Daniel Cordes, Albrecht Kadlec, Raimund Kirner, Jens Knoop, Paul Lokuciejewski, Nicholas Merriam, Marianne De Michiel, Adrian Prantl, Bernhard Rieder, Christine Rochange, Pascal Sainrat, and Markus Schordan.
Wcet 2008 - report from the tool challenge 2008.
In Proceedings of the 8th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis (WCET'08), Prague, Czech Republic, July 2008.

[19] Benedikt Huber and Martin Schoeberl.
Comparison of Implicit Path Enumeration and Model Checking Based WCET Analysis.
In Proceedings of the 9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis (WCET'09), Dublin, Ireland, July 2009.

[20] K. G. Larsen, P. Pettersson, and W. Yi.
UPPAAL **in a Nutshell.**
Journal of Software Tools for Technology Transfer (STTT), 1(1-2):134–152, 1997.

[Tarjan, 1979] Thomas Lengauer and Robert Endre Tarjan.
A fast algorithm for finding dominators in a flowgraph.
ACM Trans. Program. Lang. Syst., 1(1):121–141, 1979.

[21] Xianfeng Li, Yun Liang, Tulika Mitra, and Abhik Roychoudhury.
Chronos: A Timing Analyzer for Embedded Software.
Science of Computer Programming, 69(1–3), 2007.
Special Issue on Experimental Software and Toolkit.

References V

[22] Mingsong Lv, Wang Yi, Nan Guan, and Ge Yu.
Combining Abstract Interpretation with Model Checking for Timing Analysis of Multicore Software.
In 31st IEEE Real-Time Systems Symposium (RTSS'2010), pages 339–349. IEEE Comp. Soc., 2010.

[Benchmarks, Mälardalen Univ.] Mälardalen WCET Research Group.
WCET Project – Benchmarks.
http://www.mrtc.mdh.se/projects/wcet/benchmarks.html.

[23] Alexander Metzner.
Why model checking can improve wcet analysis.
In Rajeev Alur and Doron Peled, editors, CAV, volume 3114 of LNCS, pages 334–347. Springer, 2004.

[24] A. Prantl, M. Schordan, and J. Knoop.
TuBound – A Conceptually New Tool for WCET Analysis.
In Proceedings of the 8th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis (WCET'08), Prague, Czech Republic, July 2008.

[25] Rapita Systems Ltd.
Rapita Systems for timing analysis of real-time embedded systems.
http://www.rapitasystems.com/.

References VI

[26] B. Rieder, P. Puschner, and I. Wenzel.
Using Model Checking to Derive Loop Bounds of General Loops within ANSI-C Applications for Measurement Based WCET Analysis.
In Proc. of the 6th Int. Workshop on Intelligent Solutions in Embedded Systems (WISES'08), Regensburg, Germany, 2008.

[27] Tidorum Ltd.
Bound-T time and stack analyser.
http://www.tidorum.fi/bound-t/.

[Weiser, 1984] Mark Weiser.
Program slicing.
IEEE Trans. Software Eng., 10(4):352–357, 1984.

[28] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström.
The Worst-Case Execution-Time Problem - Overview of Methods and Survey of Tools.
ACM Trans. Embedded Comput. Syst., 7(3), 2008.