WCET Analysis

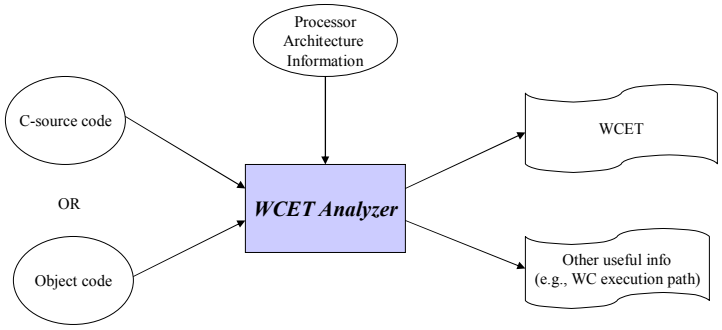
Overview

- Measurement method vs. Analytical method
- Simple analytical method
- Architecture-aware analysis

Why analytical methods?

- Measurement
  - does NOT guarantee WCET
    - The execution time depends on branches taken, loop count, cache hit/miss, etc.
    - Measurement cannot explore all possible cases
  - is very expensive and timing consuming
  - is possible only after executable code and execution environment has been setup, NOT early design stage
- Analysis
  - Guarantee WCET (safe bound)
  - Most of analysis steps can be automated (easy to use)
  - Can be used in early design stage to check 1) a piece of code is short enough, 2) interrupt handlers finish quickly enough, 3) which piece of code should be further optimized, 4) system will be schedulable, and much more

WCET Analyzer



WCET Analyzer will become an important component of IDE (Integrated Development Environment) for Embedded Real-Time Systems (along with Compiler, Debugger, RTOS, RT-monitor, etc)

Target Processors

Chip Category	Number Sold
Embedded 4-bit	2000 million
Embedded 8-bit	4700 million
Embedded 16-bit	700 million
Embedded 32-bit	400 million
DSP	600 million
Desktop 32/64-bit	150 million

1999 World Market for Microprocessors David Tennenhouse (Intel Director of Research) Keynote Speech at RTSS'99

Chip Family	Number Sold
ARM	151 million
Motorola 68k	94 million
MIPS	57 million
Hitachi SuperH	33 million
x86	29 million
PowerPC	10 million
Intel i960	7.5 million
SPARC	2.5 million
AMD 29k	2 million
Motorola M-Core	1.1 million

1999 32-bit Microcontroller Sales Microprocessor Report, Jan. 17, 2000

- Embedded Market is really big
- Developers choose the minimum processor that meets the requirement
- Most of popular embedded processor has simple architecture (RISC) – analytical method practical!
- Simple pipeline and no cache (or small on-chip cache)

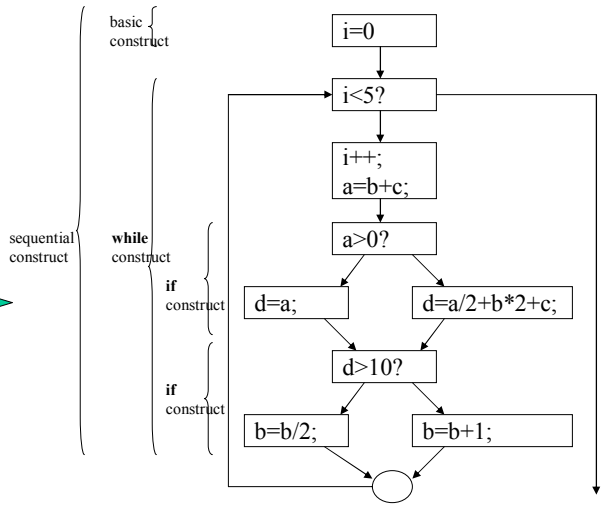
Target Software

- C, C++, and Assembly languages are most popular in embedded system market
  - Ada and Java have some use, but need for speed, small code size, and efficient access to hardware will prevent them from being dominant languages
- So, main focus is on analyzing programs written in C, C++, and Assembly languages

Basic Analysis Method

```
i=0;
while(i<5){
   i++;
   a = b+c;
   if (a >0)
         d = a;
   else
         d = a/2 + b*2 + c;
   if(d >10)
         b = b/2;
   else
         b = b+1;
}
```

Compiler generates control-flow graph

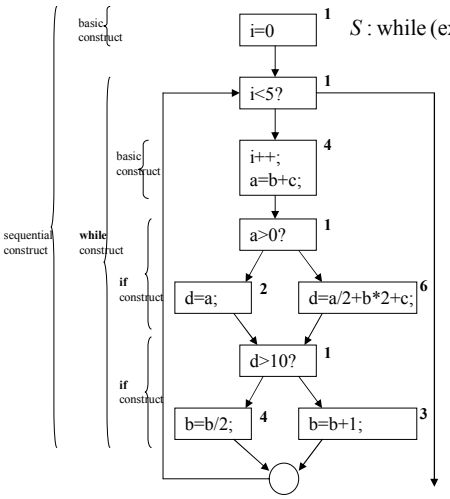


Timing Schema

$S : S_1; S_2 \Rightarrow T(S) = T(S_1) + T(S_2)$

S:if (exp) then S_1 else $S_2 \Rightarrow T(S) = T(\text{exp}) + \max(T(S_1), T(S_2))$

$S : \text{while (exp) } S_1 \Rightarrow T(S) = \sum_{i=1}^{N} (T(\text{exp}) + T(S_1)) + T(\text{exp})$



Bottom-up applying timing schema

T(1st_if) = 1+max(2,6) = 7

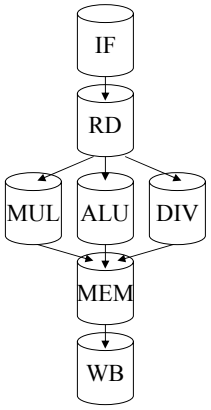
T(2nd_if) = 1+max(4,3) = 5

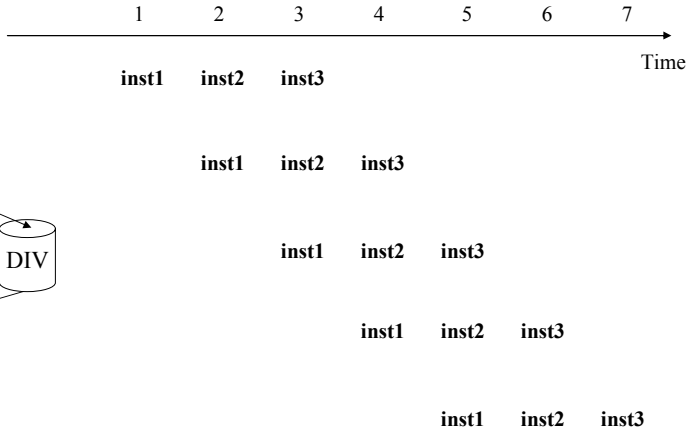
T(insideWhile) = 4+7+5=16

T(while) = 5*(1+16)+1=86

T(all)=1+86=87

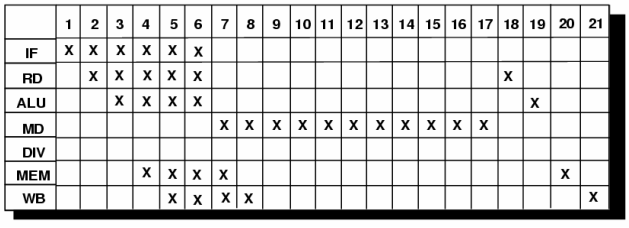
Pipelined Processor





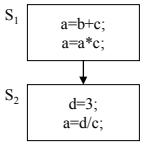
Timing Abstraction for pipeline

```
lw    $25, 16($23)
nop
lw    $24, 16($22)
nop
mult  $25,  $24
mflo  $25
```

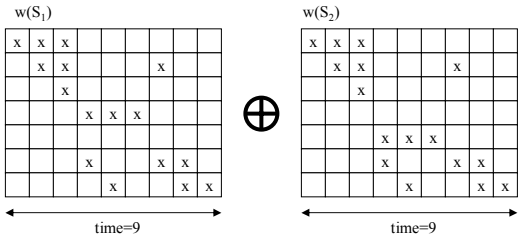


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
IF	X	X	X	X	X	X															
RD		X	X	X	X	X												X			
ALU			X	X	X	X													X		
MD							X	X	X	X	X	X	X	X	X	X	X				
DIV																					
MEM				X	X	X	X													X	
WB					X	X	X	X													X

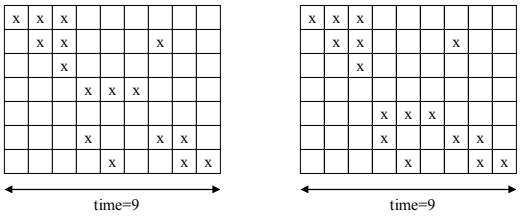
Bottom-up concatenation



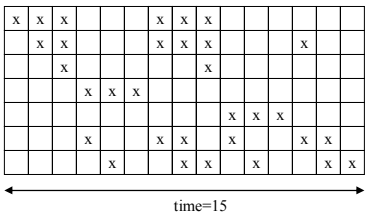
$$S : S_1; S_2 \Rightarrow w(S) = w(S_1) \oplus w(S_2)$$



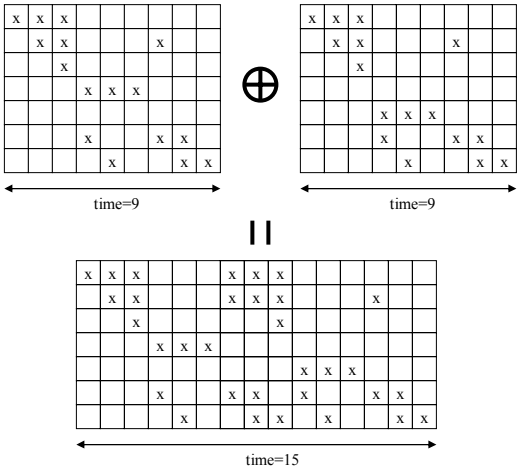
Bottom-up concatenation



=

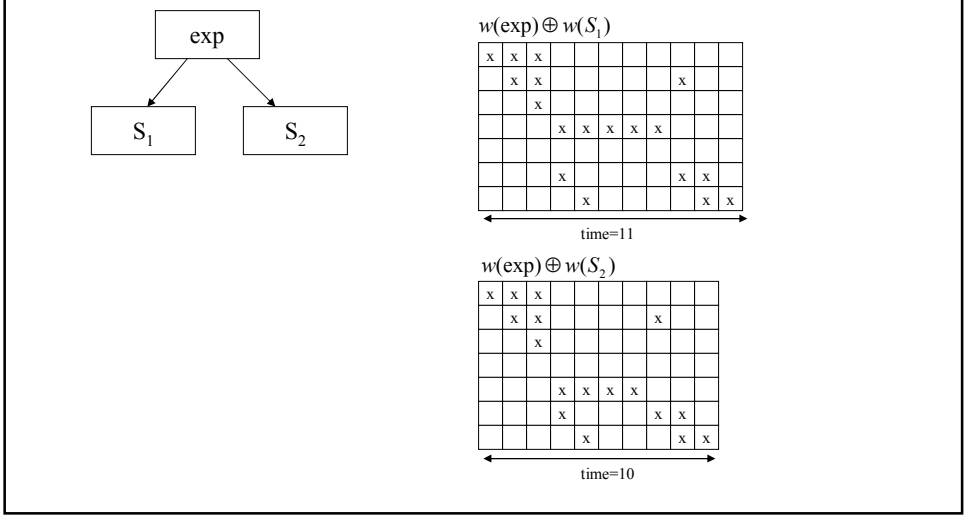


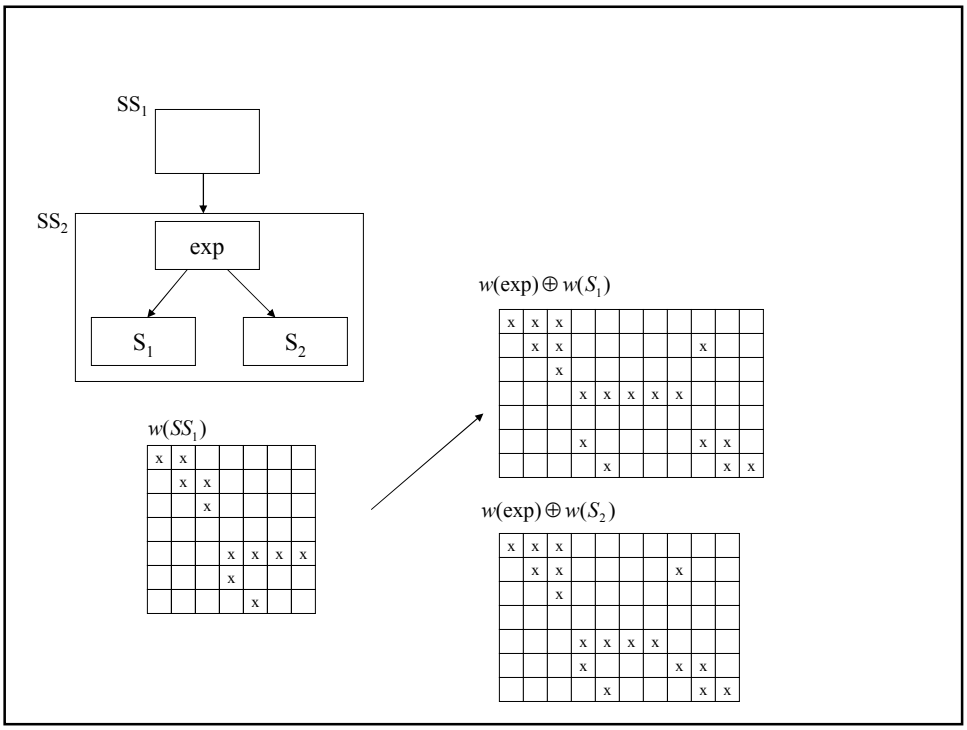
Bottom-up concatenation

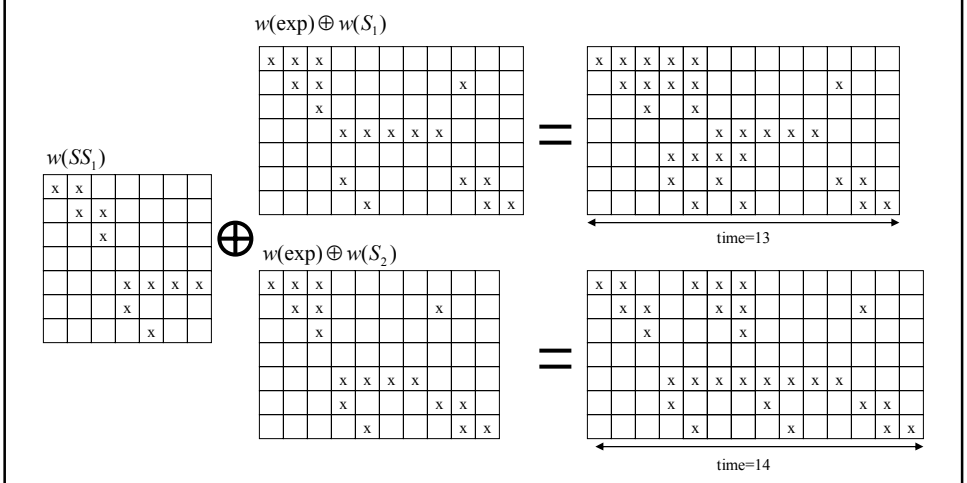


Bottom-up concatenation

- OK. Instead of just time T(S) for a construct S, we will keep the reservation table *w*(S).
- Problem: keeping only the one with max time does not guarantee WCET at the end


exp
S1
S2
w(exp) ⊕ w(S1)
time=11
w(exp) ⊕ w(S2)
time=10


SS1
SS2
exp
S1
S2
w(SS1)
w(exp) ⊕ w(S1)
w(exp) ⊕ w(S2)



Revised Timing Schema

$S : S_1; S_2 \Rightarrow W(S) = W(S_1) \oplus W(S_2)$
where $W_1 \oplus W_2 = \{w_1 \oplus w_2 \mid w_1 \in W_1, w_2 \in W_2\}$,

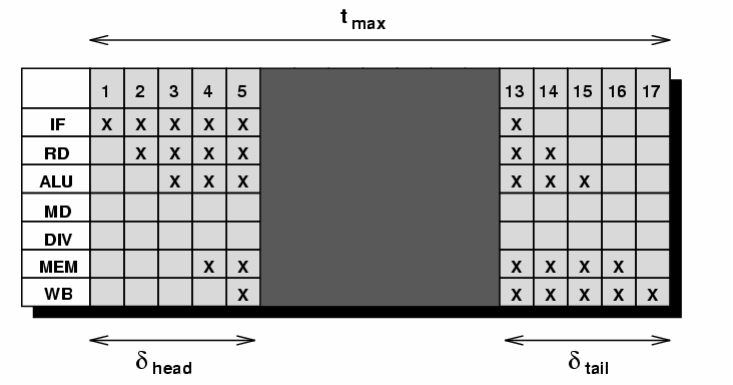
S:if (exp) then S_1 else $S_2 \Rightarrow W(S) = (W(\exp) \oplus W(S_1)) \cup (W(\exp) \oplus W(S_2))$,

S : while (exp) $S_1 \Rightarrow W(S) = \left(\bigoplus_{i=1}^{N} (W(\exp) \oplus W(S_1)) \right) \oplus W(\exp)$.

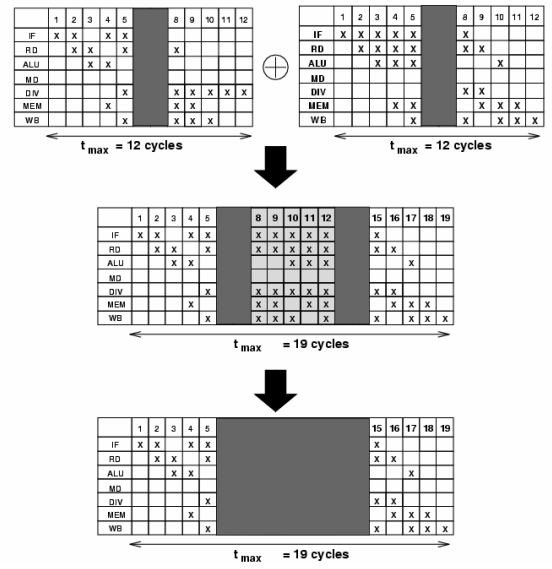
What are problems?

- Reservation tables become longer and longer as we do concatenation.
- Number of reservation tables we have to keep exponentially increase.

Abstracted Reservation Table (1)

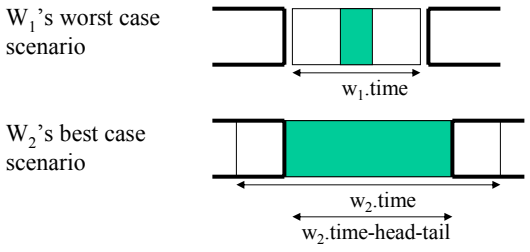


Abstracted Reservation Table (2)



Pruning

- If we are sure that a reservation table w cannot contribute to WCET, we can drop it.
- Pruning condition: if w_1's worst case scenario is better than w_2's best case scenario, then we can prune w_1.

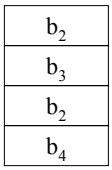


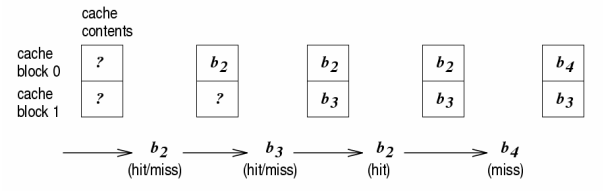
$$\exists w_2 \in W, w_1.time < w_2.time - \delta_{head} - \delta_{tail}$$

Instruction Cache Effects

- If there is on-chip instruction cache, ignoring it will give us too pessimistic WCET
- Cache hit/miss depends on previous execution path taken
- In the stage of bottom-level construct analysis, we have no idea of preceding execution path

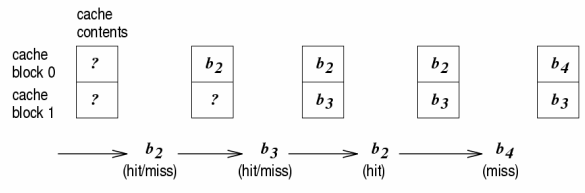
Memory reference from
a program construct S

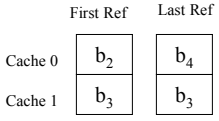




Idea

- For memory references whose cache hit/miss can be known by local analysis, accurately consider it in *w(S).time* estimation
- For other references, assume cache miss in *w(S).time* estimation
- Such pessimistic assumption will be revised in later concatenation, if the reference turns out to be cache hit





Concatenation

S_1: $b_2 \rightarrow b_3 \rightarrow b_2 \rightarrow b_4$

$w(S_1)$

	First Ref	Last Ref
Cache 0	b_2	b_4
Cache 1	b_3	b_3

$\oplus$

S_2: $b_4 \rightarrow b_2 \rightarrow b_2 \rightarrow b_8$

$w(S_2)$

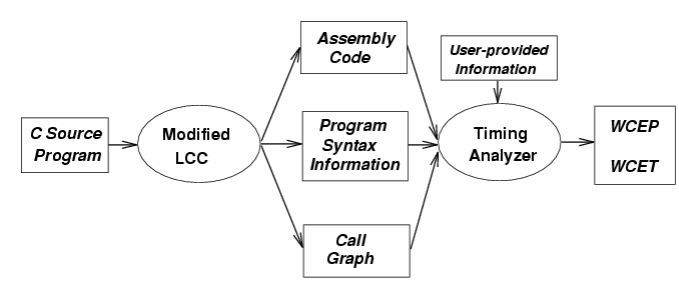
	First Ref	Last Ref
Cache 0	b_4	b_8
Cache 1	NA	NA

=

$S:S_1;S_2$ One miss counted in S_2 turns out to be hit!

	First Ref	Last Ref
Cache 0	b_2	b_8
Cache 1	b_3	b_3

WCET Analyzer



WCET analysis accuracy

	Clock	*Sort*	*MM*
Predicted	3202	14259	8376
Measured	2768	11471	6346

(unit: machine cycles)

Summary

- Commercial WCET analysis tool will come out soon (AbsInt 2004 – Reinhard Wilhelm, Saarland Unv. Germany)
  - Retargetable WCET anaysis
  - WCET anaysis for super-scalar and VLIW architecture for DSP
  - Modular implementation of WCET analysis steps
  - Integration with other development tools such as compiler, source-level debugger, RTOS, real-time kernel monitor
- References
  - Sung-Soo Lim et al., An Accurate Worst Case Timing Analysis for RISC Processors, IEEE Transactions on Software Engineering, Vol. 21, No. 7, July 1995.
  - Jakob Engblom et al., Worst-Case Execution-Time Analysis for Embedded Real-Time Systems, Journal of Software Tools for Technology Transfer, 2001.