Static Checking of Interrupt-driven Software

Jens Palsberg

Purdue University

CERIAS and Department of Computer Science

www.cs.purdue.edu/people/palsberg

Joint work with Dennis Brylow and Niels Damgaard. Supported by an NSF CAREER award.
Application Domain

Examples:

- Palm Pilots
- Cell phones
- Microcontrollers

Interrupt-driven software!
Example Program in Z86 Assembly Language

; Constant Pool (Symbol Table).
; Bit Flags for IMR and IRQ.
IRQ0  .EQU  #00000001b
; Bit Flags for external devices
; on Port 0 and Port 3.
DEV2  .EQU  #00010000b

; Interrupt Vectors.
   .ORG  %00h
   .WORD  #HANDLER  ; Device 0

; Main Program Code.
   .ORG  0Ch
   INIT:     ; Initialization section.
  0C  LD  SPL, #0F0h  ; Initialize Stack Pointer.
  0F  LD  RP, #10h   ; Work in register bank 1.
  12  LD  P2M, #00h  ; Set Port 2 lines to
                     ; all outputs.
  15  LD  IRQ, #00h  ; Clear IRQ.
  18  LD  IMR, #IRQ0
  1B  EI           ; Enable Interrupt 0.
Example Program in Z86 Assembly Language

START: ; Start of main program loop.
1C  DJNZ r2, START ; If our counter expires,
1E  LD  r1, P3 ; send this sensor’s reading
20  CALL SEND ; to the output device.
23  JP   START

SEND: ; Send Data to Device 2.
26  PUSH IMR ; Remember what IMR was.

DELAY:
28  DI ; Musn’t be interrupted
 ; during pulse.
29  LD  P0, #DEV2 ; Select control line
 ; for Device 2.
2C  DJNZ r3, DELAY ; Short delay.
2E  CLR  P0
30  POP  IMR ; Reactivate interrupts.
32  RET

HANDLER: ; Interrupt for Device 0.
33  LD  r2, #00h ; Reset counter in main loop.
35  CALL SEND
38  IRET ; Interrupt Handler is done.
.END
Our Tool

- Stack-Size Analysis

- Type Checking of Stack Elements

- Interrupt-Latency Analysis
Key Question

How much of a Z86-machine state should be represented in a flow-graph node?
One Extreme

A node contains the who Z86-machine state.

Worst case: $2^{256 \times 8} = 2^{2048}$ nodes.
The Interrupt Mask Register (IMR)

Consists of:

- A master bit (when off, all interrupts are turned off).
- One bit for each of the six interrupts.
Key Question

Can modeling just the PC and the IMR lead to a useful programming tool?
### Instructions and the corresponding edge labels

<table>
<thead>
<tr>
<th>instruction format</th>
<th>edge label</th>
<th>computation step</th>
</tr>
</thead>
<tbody>
<tr>
<td>(various)</td>
<td>e</td>
<td>no change to the stack</td>
</tr>
<tr>
<td>PUSH IMR</td>
<td>!1 (IMR value)</td>
<td>the value of the IMR is placed on the stack</td>
</tr>
<tr>
<td>PUSH (not IMR)</td>
<td>!1 “unk”</td>
<td>some value (not the IMR) is placed on the stack</td>
</tr>
<tr>
<td>CALL (label)</td>
<td>!2 (ret. addr.)</td>
<td>procedure call</td>
</tr>
<tr>
<td>(interrupt call)</td>
<td>!3 (stat. reg., ret. addr.)</td>
<td>implicit interrupt call</td>
</tr>
<tr>
<td>POP IMR</td>
<td>?1</td>
<td>the IMR is assigned the value on the top of the stack</td>
</tr>
<tr>
<td>POP (not IMR)</td>
<td>?1 “unk”</td>
<td>some register (not the IMR) is assigned the value on top of the stack</td>
</tr>
<tr>
<td>RET</td>
<td>?2</td>
<td>return from procedure call</td>
</tr>
<tr>
<td>IRET</td>
<td>?3</td>
<td>return from an interrupt handler.</td>
</tr>
</tbody>
</table>
Pushing and Popping

!1 (IMR)

---

POP IMR

m e p

?1

m e p

n e q
Assumptions

Only direct manipulation the IMR, IRQ, and SP registers.

**IMR:** We only allow IMR values to be pushed on the stack, popped from the stack, or manipulated by any binary operation in which one operand is a numeric constant, and the other is the IMR.

**IRQ:** We assume that the IRQ is read only.

**SP:** We only allow the SP to be manipulated implicitly by stack-specific instructions or by an initialization instruction.
**Interrupt-Latency Analysis**

HPI = highest-priority interrupt.

H = a predicate which is true of a node $n$ iff the PC component of $n$ is the start address of the handler for the HPI.

\[\begin{align*}
Red & \equiv \text{it is not possible to reach a node in } H \\
Yellow & \equiv \neg(\text{Red} \lor \text{Green}) \\
Green & \equiv \text{it is inevitable that computation will reach a node in } H.
\end{align*}\]

**Key Observation:** If an HPI is pending when computation reaches a node $n$, and there is an edge from $n$ to a node in $H$, then computation will proceed along such an edge.
Interrupt-Latency Analysis

\( \text{UltraGreen} \equiv \) there is an edge to a node in \( H \).

\( \text{Green} \equiv \) a node in \( H \), or
it is inevitable that computation
will reach an ultragreen node.

In Computation Tree Logic (CTL), we can make the intuition precise by
defining the colors as predicates on nodes:

\[
\begin{align*}
\text{Red} & \equiv \neg \text{EF}(H) \\
\text{Yellow} & \equiv \neg (\text{Red} \lor \text{Green}) \\
\text{UltraGreen} & \equiv \text{EX}(H) \\
\text{Green} & \equiv H \lor \text{AF}(\text{UltraGreen}).
\end{align*}
\]

Given a flow graph \( G \) and a formula \( \phi \) in CTL, it can be decided
in \( O(|G| \times |\phi|) \) time whether \( \phi \) is true or false of the nodes in \( G \).
## Measurements

<table>
<thead>
<tr>
<th>Program</th>
<th>Nodes</th>
<th>Edges</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTurk</td>
<td>1,209</td>
<td>2,316</td>
<td>4.01 s</td>
<td>31.6 MB</td>
</tr>
<tr>
<td>GTurk</td>
<td>1,581</td>
<td>3,101</td>
<td>4.20 s</td>
<td>32.2 MB</td>
</tr>
<tr>
<td>ZTurk</td>
<td>1,493</td>
<td>2,885</td>
<td>4.12 s</td>
<td>32.1 MB</td>
</tr>
<tr>
<td>DRop</td>
<td>1,138</td>
<td>2,043</td>
<td>4.02 s</td>
<td>31.1 MB</td>
</tr>
<tr>
<td>Rop</td>
<td>1,217</td>
<td>2,278</td>
<td>4.08 s</td>
<td>31.7 MB</td>
</tr>
<tr>
<td>Fan</td>
<td>5,149</td>
<td>17,195</td>
<td>5.13 s</td>
<td>39.3 MB</td>
</tr>
<tr>
<td>Serial</td>
<td>394</td>
<td>1,082</td>
<td>3.78 s</td>
<td>31.0 MB</td>
</tr>
<tr>
<td>Example</td>
<td>148</td>
<td>222</td>
<td>3.16 s</td>
<td>34.9 MB</td>
</tr>
</tbody>
</table>
# Measurements

<table>
<thead>
<tr>
<th>Program</th>
<th>Lower</th>
<th>Upper</th>
<th>Time</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTurk</td>
<td>17</td>
<td>18</td>
<td>4.11 s</td>
<td>31.6 MB</td>
</tr>
<tr>
<td>GTurk</td>
<td>16</td>
<td>17</td>
<td>4.31 s</td>
<td>32.2 MB</td>
</tr>
<tr>
<td>ZTurk</td>
<td>16</td>
<td>17</td>
<td>4.22 s</td>
<td>32.1 MB</td>
</tr>
<tr>
<td>DRop</td>
<td>12</td>
<td>14</td>
<td>4.14 s</td>
<td>31.1 MB</td>
</tr>
<tr>
<td>Rop</td>
<td>12</td>
<td>14</td>
<td>4.18 s</td>
<td>31.8 MB</td>
</tr>
<tr>
<td>Fan</td>
<td>11</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Serial</td>
<td>10</td>
<td>10</td>
<td>3.87 s</td>
<td>31.0 MB</td>
</tr>
<tr>
<td>Example</td>
<td>37</td>
<td>37</td>
<td>3.21 s</td>
<td>34.9 MB</td>
</tr>
</tbody>
</table>

The lower bounds were found with a software simulator for Z86 assembly language that we wrote.
# Measurements

## Interrupt latency analysis of highest priority IRQ

<table>
<thead>
<tr>
<th>Program</th>
<th>Ultragreen</th>
<th>Green</th>
<th>Ultrayellow</th>
<th>Yellow</th>
<th>Red</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTurk</td>
<td>43%</td>
<td>51%</td>
<td>34%</td>
<td>49%</td>
<td>0%</td>
<td>260</td>
</tr>
<tr>
<td>GTurk</td>
<td>43%</td>
<td>50%</td>
<td>30%</td>
<td>50%</td>
<td>0%</td>
<td>272</td>
</tr>
<tr>
<td>ZTurk</td>
<td>42%</td>
<td>50%</td>
<td>30%</td>
<td>50%</td>
<td>0%</td>
<td>276</td>
</tr>
<tr>
<td>DRop</td>
<td>15%</td>
<td>19%</td>
<td>60%</td>
<td>81%</td>
<td>0%</td>
<td>312</td>
</tr>
<tr>
<td>Rop</td>
<td>15%</td>
<td>19%</td>
<td>58%</td>
<td>81%</td>
<td>0%</td>
<td>312</td>
</tr>
<tr>
<td>Fan</td>
<td>56%</td>
<td>67%</td>
<td>24%</td>
<td>33%</td>
<td>0%</td>
<td>310</td>
</tr>
<tr>
<td>Serial</td>
<td>43%</td>
<td>79%</td>
<td>14%</td>
<td>21%</td>
<td>0%</td>
<td>326</td>
</tr>
<tr>
<td>Example</td>
<td>25%</td>
<td>46%</td>
<td>30%</td>
<td>54%</td>
<td>0%</td>
<td>242</td>
</tr>
</tbody>
</table>

Latencies are given in machine cycles.

One machine cycle is executed in 1 microsecond.
Conclusion

- Modeling PC+IMR gives a good stack-size analysis, a good type checker, and a reasonable interrupt-latency analysis.

- Our tool is one of the first to give an efficient and useful static analysis of assembly code.
And the work continues

- Identification of loop variables, to get rid of yellow nodes.

- Typed assembly language.

- Motorola 68000-family processors.