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: $2^{256*8} = 2^{2048}$ nodes.

The Interrupt Mask Register (IMR)

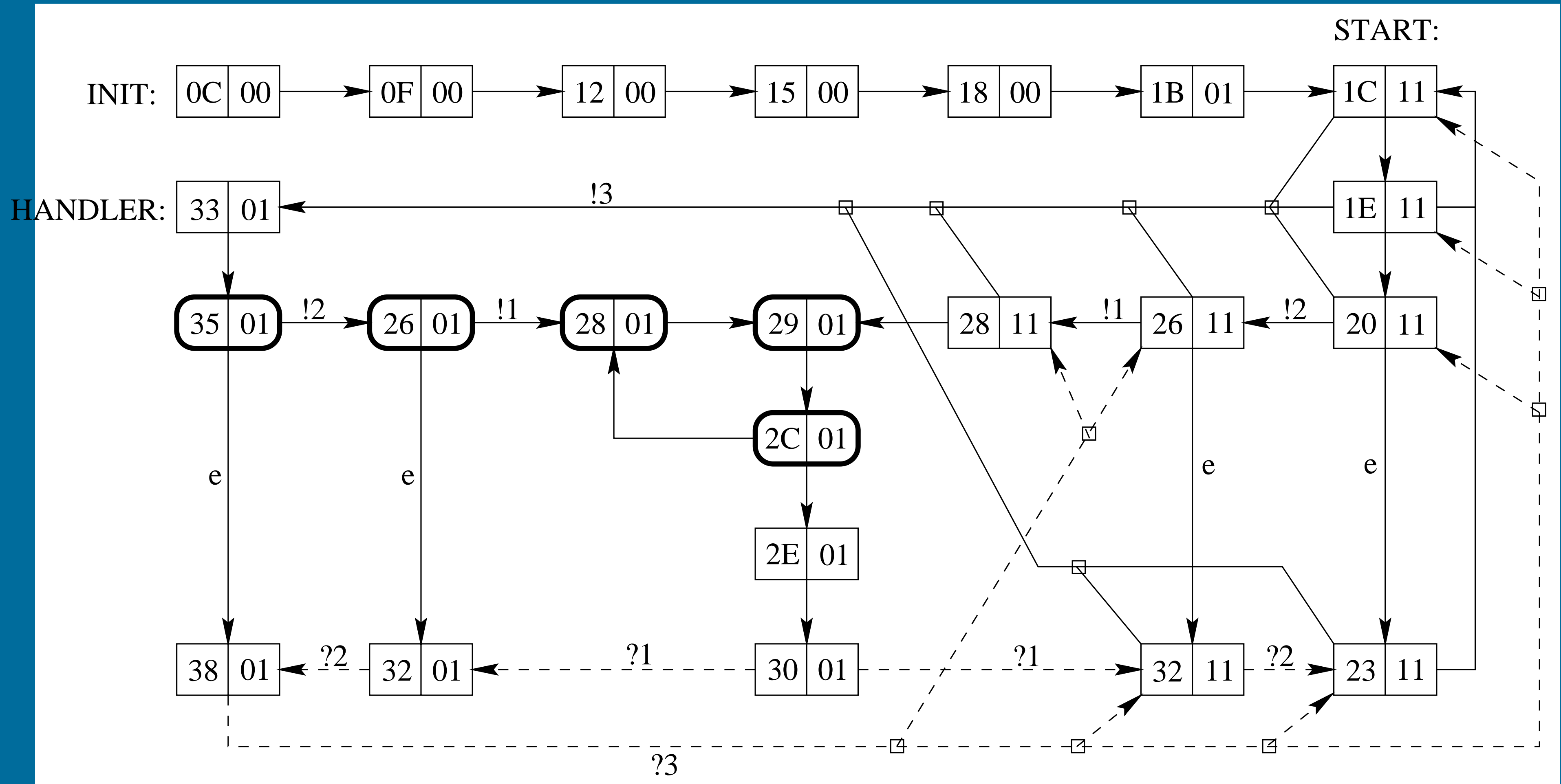
Consists of:

- 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?

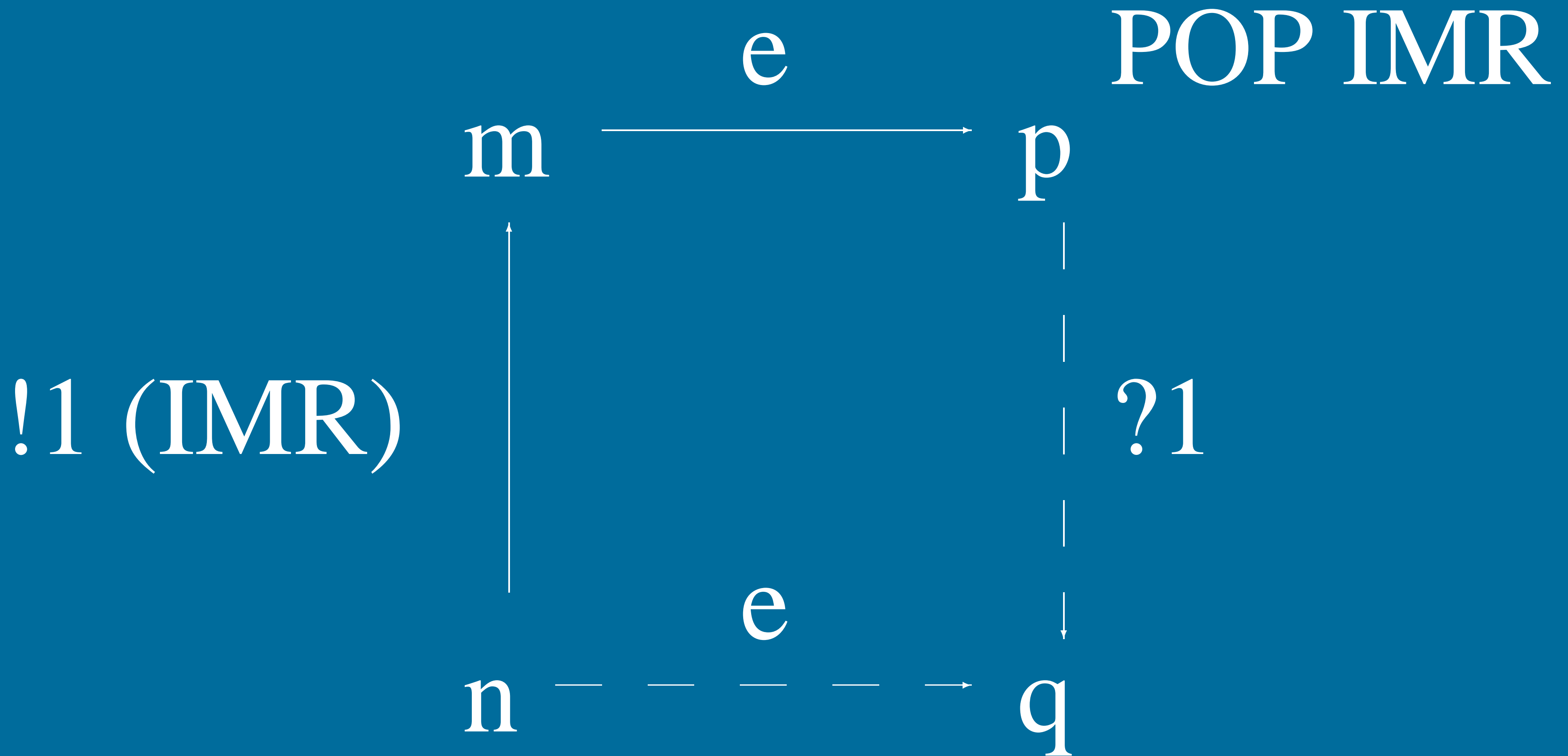
Flow Graph



Instructions and the corresponding edge labels

instruction format	edge label	computation step
⟨various⟩	e	no change to the stack
PUSH IMR	!1 (IMR value)	the value of the IMR is placed on the stack
PUSH ⟨not IMR⟩	!1 “unk”	some value (not the IMR) is placed on the stack
CALL ⟨label⟩	!2 (ret. addr.)	procedure call
⟨interrupt call⟩	!3 (stat. reg., ret. addr.)	implicit interrupt call
POP IMR	?1	the IMR is assigned the value on the top of the the stack
POP ⟨not IMR⟩	?1 “unk”	some register (not the IMR) is assigned the value on top of the stack
RET	?2	return from procedure call
IRET	?3	return from an interrupt handler.

Pushing and Popping



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.

Red $\equiv$ it is not possible to reach a node in H
$Yellow$ $\equiv$ $\neg(Red \vee Green)$
$Green$ $\equiv$ it is inevitable that computation will reach a node in H.

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

$$
\begin{aligned}
\mathit{UltraGreen} &\equiv \text{there is an edge to a node in H.} \\
\mathit{Green} &\equiv \text{a node in H, or} \\
&\ \text{it is inevitable that computation} \\
&\ \text{will reach an ultragreen node.}
\end{aligned}
$$

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

$$
\begin{aligned}
\mathit{Red} &\equiv \neg\, \mathsf{EF}(\mathsf{H}) \\
\mathit{Yellow} &\equiv \neg(\mathit{Red} \vee \mathit{Green}) \\
\mathit{UltraGreen} &\equiv \mathsf{EX}(\mathsf{H}) \\
\mathit{Green} &\equiv \mathsf{H} \vee \mathsf{AF}(\mathit{UltraGreen}).
\end{aligned}
$$

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

Measurements

Building the graph				
Program	Nodes	Edges	Time	Space
CTurk	1,209	2,316	4.01 s	31.6 MB
GTurk	1,581	3,101	4.20 s	32.2 MB
ZTurk	1,493	2,885	4.12 s	32.1 MB
DRop	1,138	2,043	4.02 s	31.1 MB
Rop	1,217	2,278	4.08 s	31.7 MB
Fan	5,149	17,195	5.13 s	39.3 MB
Serial	394	1,082	3.78 s	31.0 MB
Example	148	222	3.16 s	34.9 MB

Measurements

Stack-size analysis				
Program	Lower	Upper	Time	Space
CTurk	17	18	4.11 s	31.6 MB
GTurk	16	17	4.31 s	32.2 MB
ZTurk	16	17	4.22 s	32.1 MB
DRop	12	14	4.14 s	31.1 MB
Rop	12	14	4.18 s	31.8 MB
Fan	11	N/A	N/A	N/A
Serial	10	10	3.87 s	31.0 MB
Example	37	37	3.21 s	34.9 MB

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

Measurements

Interrupt latency analysis of highest priority IRQ						
Program	Ultragreen	Green	Ultrayellow	Yellow	Red	Latency
CTurk	43%	51%	34%	49%	0%	260
GTurk	43%	50%	30%	50%	0%	272
ZTurk	42%	50%	30%	50%	0%	276
DRop	15%	19%	60%	81%	0%	312
Rop	15%	19%	58%	81%	0%	312
Fan	56%	67%	24%	33%	0%	310
Serial	43%	79%	14%	21%	0%	326
Example	25%	46%	30%	54%	0%	242

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.