“Experience is what you get...
...when you don't get what you want.”

Debugging
Oct. 27, 2011

Dave Eckhardt

Debugging

As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs.

– Maurice Wilkes (1949)

Outline

What is “Debugging”?

Programming languages

- **Whitespace**
- **INTERCAL**
- **M**

A debugging story

Conclusions

What is “Debugging”?

Debugging is resolving a clash between stories

- Your hopeful story of achievement
- The world's sad tale of woe

The stories look alike!

- At the beginning, they both start with main()...
- Key step: finding the divergence

Stories are fractal

- You can zoom in on them and get more detail each time
- The divergence is typically a tiny detail
  - You will need to zoom in quite a lot

A Whitespace Program

"Count from 1 to 10" (partial listing)

Features of Whitespace

- Only space, tab, and line-feed encode program statements
- All other characters (A-Z, a-z, 0-9, etc.) encode comments
- Simple stack-based language

Whitespace "Explained"

Statement	Meaning
[Space][Space][Space] [Tab][LF]	Push 1 onto stack
[LF][Space][Space][Space] [Tab][Space][Space][Space] [Space][Tab][Tab][LF]	Set a label at this point
[Space][LF][Space]	Duplicate the top stack item
[Tab][LF][Space][Tab]	Output the current value
...	...

INTERCAL

Features of INTERCAL

- Designed late one night in 1972 by two Princeton students
- Deliberately obfuscated language

Variables

- 16-bit integers, .1 through .65535
- 32-bit integers, :1 through :65535

Operators

- Binary: “mingle”, “select”
- Unary: AND, OR, XOR
  - How are those unary???
  - Simple: AND and's together adjacent bits in a word
- Simplest way to put 65536 in a 32-bit variable?
  - `DO :1 <- #0¢#256`

The language “M”

Features of M

- Also designed in the 1970's
- More widely used than Whitespace, INTERCAL

Variables

- 32-bit integer variables: A, B, C, D, DI, SI, S
- One array, M[]
  - Valid subscripts range from near zero to a large number
  - But most subscripts in that range will crash your program!
- A stack, located in M[], generally pointed to by S

Statements

- Lots of arithmetic and logical operations
- Input and output use a special statement called OUCH!

“C” Example

Print out numbers 1-10

```
for (i = 1; i < 10; ++i)
{
  print_int(i);
}
```

“M” Example

Print out numbers 1-10

Address	Instruction		Comment
00001000	MOV	$1,M[10]	Init loop index
00001004	MOV	$10,M[11]	Init loop limit
00001008	MOV	M[10],A	Fetch loop index
00001012	MOV	M[11],B	Fetch loop limit
00001016	COMPARE	A,B	Compare
00001020	EQUAL?	$1044	If so, done
00001024	PUSH	A	Push loop index
00001028	ADD	$1,A	Increment for next time
00001032	MOV	A,M[10]	Store index for next time
00001036	CALL	$4000	print_int()
00001040	JUMP	-32	Top of loop

Classifying Instructions

Print out numbers 1-10

Address	Instruction		Type
00001000	MOV	$1,M[10]	Store
00001004	MOV	$10,M[11]	Store
00001008	MOV	M[10],A	Fetch
00001012	MOV	M[11],B	Fetch
00001016	COMPARE	A,B	ALU
00001020	EQUAL?	$1044	Absolute branch
00001024	PUSH	A	Stack
00001028	ADD	$1,A	ALU
00001032	MOV	A,M[10]	Store
00001036	CALL	$4000	Absolute branch
00001040	JUMP	-32	Relative branch

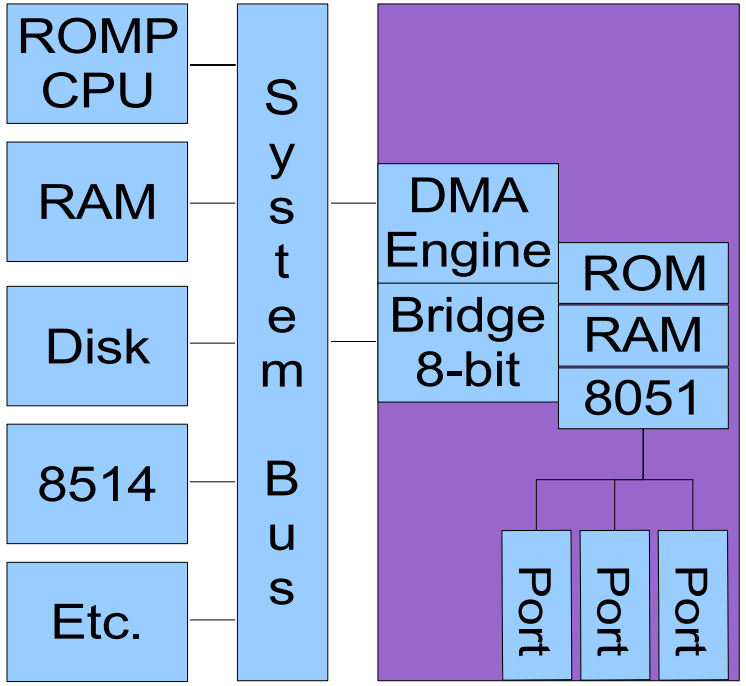
“Reserved (Set to Zero)”

The Scene

- Mid-1980's
- IBM “RT PC”
- “Multiprotocol adaptor”
  - Intel 8051
  - Some Intel bus bridge
  - DMA engine
  - RS-232/422 ports
  - Some bizarre dial port

Key features

- Narrow Intel bus bridge
- Protocol code loaded from host device driver
- Boot loader in ROM



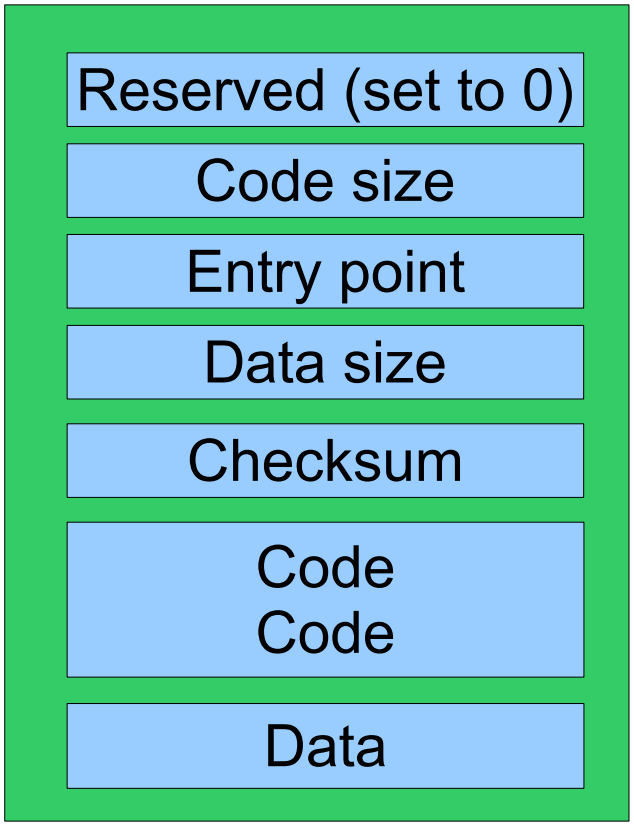
“Reserved (Set to Zero)”

Microcode

- Intel 8051 binary
- Commanded by host to transfer data across bus to/from RAM
- Able to code/decode packet data onto/from wire
- Card could implement checksum and retransmission

File format

- Code size
- Entry point
- ...



The Fun Begins

First program

- Send 1-byte constant to host port (generates interrupt)
- Enter infinite loop
- ⇒ Works!

The Fun Begins

First program

Send 1-byte constant to host port (generates interrupt)

Enter infinite loop

⇒ Works!

Second program

Program port #0 to be RS-232, no modem control, 9600 bps

Transfer packet from host via DMA

```
while (ptr < end)
  while (!IDLE(0))
    continue;
  output(*ptr++);
```

⇒ Hangs silently!

Houston, We Have A Problem

Inquiry is hard

- No way to inspect card – RAM, registers, etc.
  - *Everything* is under control of boot loader or downloaded code
    - » Code wedges ⇒ no more data
- Only two forms of communication possible
  - 8-bit code sent by bridge with host interrupt
    - » Simple enough: out(port, val)
    - » Only 8 bits (more like 7)
  - DMA
    - » Can send arbitrary data dumps
    - » Except it doesn't work

#include <laborious_debugging_session.h>

Time Passes...

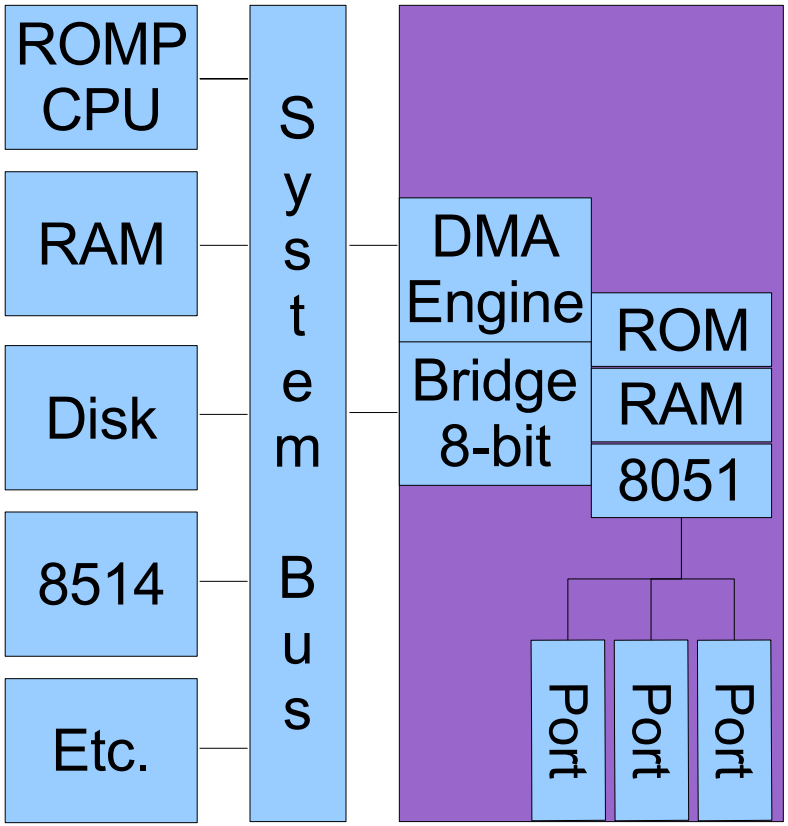
Basic approach

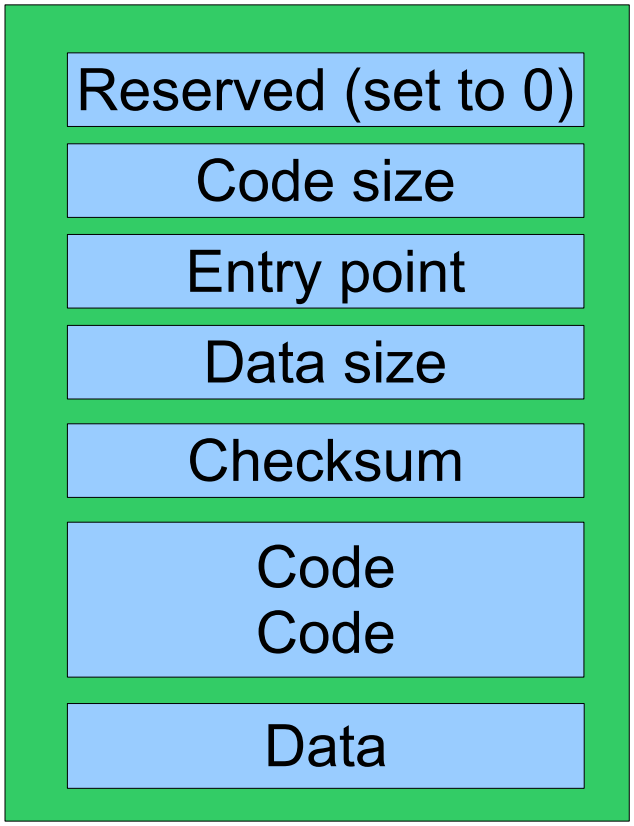
- Write a tiny program
- Download it
- See what tiny answer it returns (if any)

Results

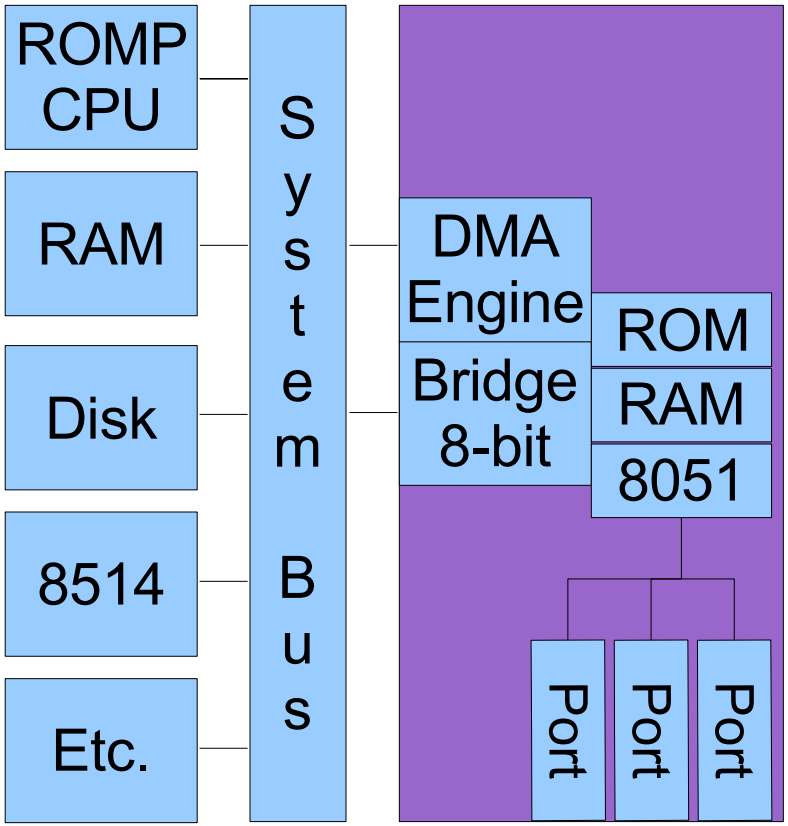
- *Most* 8051 instructions appear to work
  - Port input/port output (thankfully!)
  - Arithmetic, shift/roll
  - RAM load/store
  - Relative branch
- Some instructions don't work so well
  - Call/return
  - Jump (absolute branch)
- ??? (????)

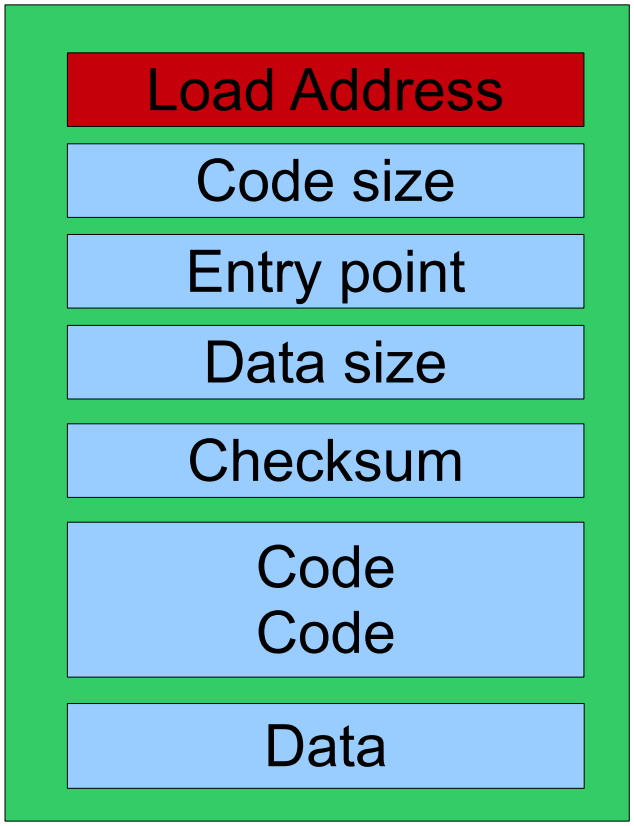
Food for Thought?





“Reserved (Set to Zero)”!!!!!





Residual Amusement

Contacting the developers inside IBM

- Very hard
  - Intentionally very hard
- Insistence
  - That's not how it works!
  - The documentation is correct!
  - We know how our product works!

Residual Amusement

Contacting the developers inside IBM

- Very hard
  - Intentionally very hard
- Insistence
  - That's not how it works!
  - The documentation is correct!
  - We know how our product works!
- Further insistence
  - “Steve” no longer works in the group
    - » After some time, his notes turn up
    - » “It appears that the development version did that”
    - » “No card with that behavior ever shipped to customers”

Lessons?

Is this just a horror story?

- Are there lessons?

Observations

- The problem wasn't “in the code”
  - All programs downloaded to the card were correct
    - » Well, most
  - The boot loader was also correct
  - But the execution environment was (subtly) wrong
    - » printf()/gdb approach would address situation only diffusely
    - » ...if available (not even close!)

What is debugging really?

Debugging

Two stories

- Plan: Hopes and dreams for the future of humanity
- Observation: Tale of woe

Key observation

- They are mostly the same story!
  - The beginnings are *identical*
  - Somewhere there is a tiny discrepancy
  - The stories continue “in a similar vein” after the divergence
  - One story ends in disaster

Debugging

How to progress?

- A *deep* understanding of the stories is necessary
  - Branch vs. Jump!
  - All abstractions are gone at that point
- Measuring which parts happen correctly is *not easy*
  - May require embedding test code in application
  - May require writing entire test applications

This looks like science!

- Hypothesis
- Experimental design
- Measurement
- Analysis

Debugging Suggestions

Move beyond "plot summaries"

- "My program dies"
- "My program dies with a segmentation fault"
- "My program dies with a segmentation fault in xxx()"
  - That one is getting *closer*, but...

Deepen your level of detail

- What was your story of hope, in detail?
- What parts of your story *already happened*?

Measurement Techniques

"Obvious"

- printf()
- single-step the program

Moving beyond the obvious

- Know your debugger
  - breakpoints, watchpoints
- Those pesky registers (in assembly code)
  - The values should always make sense – all of them

Writing code

- Breakage of a complex data structure is, well complex
- Probably need code to check invariants
  - Doing it by hand is fun at most once

Record-Keeping

While you're working

- Keep a “bug diary”
  - What you've tried
  - What you've learned
  - What you don't know how to measure
  - What results are confusing
  - What to try next

After you find the bug

- Keep a “bug diary”
  - Last week I found an xxx bug
  - This week I found an xxx bug
  - Maybe I should check for xxx bugs when I run into trouble
- This is part of the “Personal Software Process” (PSP)

Asking for Help

- **"Plot summary" is not enough**
  - We probaby have no idea what's wrong
    - Really!
- **You should always have a measurement plan**
  - What is the next thing to measure?
  - How would I measure it?
- **You may reach the end of your rope**
  - Some things are genuinely tricky to debug
    - This is a good learning experience

Asking for Help

When are you ready to ask for help?

- You have a long, detailed story – this is *critical!!!*
  - "Story" often needs one or two pictures
- Parts of the story are clearly happening
  - You have straightforward evidence, you are confident
- You have a measurement problem
  - Too many things to measure?
  - No idea how to measure one complicated thing?
  - Measurement results "make no sense"?

Summary

Debugging is about reconciling two stories

- **“Plot summaries” aren't stories (you must zoom in)**
- **“If you don't know where you are going, you will wind up somewhere else.” — Yogi Berra**

Measure multiple things, use multiple mechanisms

You should “always” have a next measurement target

When you ask for help, bring a long story

- **...which you will naturally be an expert on the first part of**