Retro Programming

SUNDAY, 21 SEPTEMBER 2008

Optimising Assembly Like an 80's Hacker

Forget about fancy algorithms and data structures. If you want respect as an 80's hacker, follow these simple tips.

Never get caught setting a register to zero without using xor:

Z80 Code

```
ld a,0          ; bad, 2 bytes / 7 cycles

xor a           ; good, 1 byte / 4 cycles
```

8088 Code

```
mov ax,0        ; bad, 3 bytes / 4 cycles

xor ax,ax       ; good, 2 bytes / 3 cycles
```

Never set two 8 bit register independently. Code readability is not required:

Z80 Code

```
ld b,10         ; bad, 4 bytes / 14 cycles
ld c,32

ld bc,10*256+32  ; good, 3 bytes / 11 cycles
```

8088 Code

```
mov ch,10       ; bad, 4 bytes / 8 cycles
mov cl,32

mov cx,10*256+32 ; good, 3 bytes / 4 cycles
```

Never compare to zero:

Z80 Code

```
cp 0            ; bad, 2 bytes / 7 cycles

or a            ; good, 1 byte / 4 cycles
```

8088 Code

MOST POPULAR:

16-Bit Xorshift Pseudorandom Numbers in Z80 Assembly

Classic Home Computer Fonts

#songsincode - Lyrics for Programmers

Optimising Assembly Like an 80's Hacker

Itsy Forth: The Compiler

10 Alternative Monospace Fonts for Programmers

8 Bit Home Computer Benchmarks

Itsy-Forth: the Dictionary and Inner Interpreter

Fast Z80 Bit Reversal

Z80 Size Programming Challenge #4

► 2017 (4)
► 2016 (2)
► 2015 (4)
► 2014 (5)
► 2013 (4)
► 2012 (8)
► 2011 (12)
► 2010 (13)
► 2009 (18)
▼ 2008 (9)
► December (1)
► November (1)
► October (1)
▼ September (3)
Optimising Assembly Like an 80's Hacker
Al Zimmermann's Programming Contests
Five Programming Books I Couldn't Survive Without
► August (1)
► June (2)

```
cmp ax,0        ; bad, 3 bytes / 4 cycles

test ax,ax      ; good, 2 bytes / 3 cycles
```

Remember, you don't need to worry about code alignment, order of instructions or processor penalties. Follow these simple tips and your super-optimised bubble sort will demand the utmost respect!

Posted by John Metcalf
Labels: 8088, asm, assembly, optimisation, optimise, z80

36 comments:

John's Tech Blog 26 October 2008 at 02:30

Great Post!

I was sorting my office out the other day and came across my first "useful" program which was written completely in assembler! I'll be putting it on my blog in the near future....! :)

-John.

Reply

Anonymous 6 February 2009 at 09:51

I try and tell the "kids" of today about saving cycles and memory contrants, they just don't get it.

Reply

Anonymous 13 February 2009 at 03:14

But don't forget: Premature optimalization is root of all evil.

Reply

Anonymous 13 February 2009 at 03:51

So who said what to anger you into writing this post?

Reply

Harold Fowler 13 February 2009 at 04:16

Wow, hacking and phreaking in the 80s was so much fun. Now everything is a federal offense so why bother!

RT
www.anon-tools.us.tc

Reply

Anonymous 13 February 2009 at 04:19

Most time such low-level optimization is exaggerated. Rather spend more time on software design.

Reply

Paolo Bonzini 13 February 2009 at 05:26

To the guys saying "Most time such low-level optimization is exaggerated" or stuff like that, remember that the compiler _is_ using xor instead of mov and stuff like that on platforms where it matters.

Reply

rif 13 February 2009 at 05:28

Optimizing Z80 and 8088 assembler was how I spent a great deal of my time in the 1980'ies. Still very useful when you need to program single chip systems and micro controllers.

Reply

richmoore 13 February 2009 at 05:31

Another one is never have a JSR followed by a RET. Use a JMP instead saving you lots of cycles on a 6502 (and I'd guess the same with the equivalent 8086 instruction).

Reply

Ivan Koblik 13 February 2009 at 05:32

This kind of optimization doesn't require additional efforts, you just use these operations instead of the others. That's it!
Try to disassemble any C/Delphi etc. program, you won't EVER see "mov ax,0"!
You'll just get used to reading "xor ax,ax" as "ax=0".

Reply

Anonymous 13 February 2009 at 05:34

There are some who belittle this type of optimization, but they are often the same ones who produce huge bloated code that wipes out 1MB L2 caches and slows 2GHz machines to a crawl.

More a project similar in essence to your ideal, google for "fbui".

Reply

Anonymous 13 February 2009 at 06:04

@most commentators:

Didn't you read the last paragraph... or didn't you get the irony in it?

Reply

Ivan Koblik 13 February 2009 at 06:11

Well I think most of us got it.
Just, I was replying to the comments, not to the article itself ;)

Reply

Anonymous 13 February 2009 at 06:23

the same idiots who don't optimize code are the ones who cant code a hello world program in less than 1mb.

Reply

Anonymous 13 February 2009 at 06:34

What WAS the point of the last paragraph? Making fun of programmers in the eighties for not optimizing for processor features they didn't have? Making fun of programmers now for writing assembly code like it was still the eighties? Neither makes one lick of sense.

Reply

Anonymous 13 February 2009 at 06:42

Anonymous (is that your real name?), I think he was poking fun at the ridiculous of the article itself. I.e. the article contains some cool tricks but the gains tend to pale into insignificance beside the larger bottlenecks one tends to find.

Reply

Anonymous 13 February 2009 at 06:56

Don't we have optimizing compilers for these kinds of things nowadays?

Reply

Anonymous 13 February 2009 at 08:19

Optimizing Assembly is so passe...

Reply

Bill 13 February 2009 at 08:23

I remember having to do that!

Now I can usually trust any decent 'C' compiler to do most of the job and the low cost of high performance silicon to do the rest.

I recently had to code an 8051 emulator, cross platform, but optimised for MIPS. I was surprised at how much performance I gained (~50%) after examining the object code and implementing a few simple tweaks (minimal inline asm, mainly type-forcing).

It's fast enough now, but I keep looking at the code, knowing that I could get another 200% by hand coding in asm.

Also - Some may accuse me of heresy, but MSC (7 or 8) produces *much* better 80x86 code than GCC.

Reply

Anonymous 13 February 2009 at 08:33

The hell kind of 80s hacker doesn't code on a 6502?

Reply

Bill 13 February 2009 at 08:41

6502 -

Apple II - too trendy, too expensive

Commodore 64 - too WalMart, too limited

Plus the Z80 was a much better processor.

Reply

Anonymous 13 February 2009 at 09:09

8051 FTW!!!

Reply

Anonymous 13 February 2009 at 10:52

8031/8051 and 8048, the worst instruction sets of all time!

Reply

Anonymous 13 February 2009 at 10:54

Another 6502 tip - use "zero page" for frequently-access variables. One fewer clock cycle for the lookup. (50% savings! w00t!)

LDA $1001 ;2 cycle fetch
LDA $01 ;1 cycle fetch

--JT

Reply

Anonymous 13 February 2009 at 10:59

well, then 8052 FTW!!!

Reply

Anonymous 13 February 2009 at 11:16

Shall we play a game?

Reply

Anonymous 13 February 2009 at 11:27

only an 8 bit game then

Reply

Anonymous 13 February 2009 at 11:35

If you're a bit interested in compiler's source optimization: http://www.fefe.de/know-your-compiler.pdf
Gist: fast code = important; readable code = more important

Reply

Anonymous 13 February 2009 at 11:38

6809 rules. Has a direct page instead of a zero page.

Reply

Henning 13 February 2009 at 13:28

Another nice trick was to use shl/shr to multiply/divide by a power of 2.

Reply

dainanaki 13 February 2009 at 16:39

Sad thing is that I know most of these tricks but I don't have much luck keeping the stack aligned. Anyone have some advice for me?

Reply

Anonymous 19 February 2009 at 19:34

I guess a good question would be why program in assembler? When I 1st started programming, computers were 16x16 and $10,000 (Model 80), so it can't be for speed (64x since then with a good C compiler running only 2x behind). I think its for the thrill of direct control (down to the engine room).

Reply

Mikkel Alan Stokkebye Christiansen 3 December 2009 at 08:54

LD bc,nn is 10 cycles if i'm not mistaken.
Your timing for the 8088 is only the cycles used in the kernel of the cpu, not the actual cycles it takes.
0 1 2 3 4 5 6 Prefetched
16:1 12:1 8:1 4:1 4:2 4:3 4:4 MOV AX,0
11:¾ 7:¾ 3:¾ 3:1¾ 3:2¾ 3:3¾ 3:4¾ XOR AX,AX
20:1 16:1 12:1 8:1 4:1 4:2 4:3 MOV CH,10!MOV CL,32
16:1 12:1 8:1 4:1 4:2 4:3 4:4 MOV CX,10*256+32
16:1 12:1 8:1 4:1 4:2 4:3 4:4 CMP AX,0
11:¾ 7:¾ 3:¾ 3:1¾ 3:2¾ 3:3¾ 3:4¾ TEST AX,AX

The number after the : is the number of bytes in the prefetch queue. ¾ means, that it will take 1 cycle more to get the next byte read.

Reply

Anonymous 22 April 2011 at 07:07

6809 was the best designed 8-bit processor every made!

When writing ROM code for the Tandy CoCo, a great optimization was:
PUSH A
PUSH B
... rest of subroutine ...
PULS A,B,PC
register restore + RTS in 1 instruction!

Reply

Anonymous 18 April 2012 at 12:44

Also, the 6809 is a great Forth processor as it has two stack pointers.

Reply

xcod4r 14 January 2019 at 07:22

assembly programming on assembly exampleswe created assembly code site

Reply

Enter your comment...

Comment as: Stephen Justin ▾ Sign out

Publish Preview Notify me

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)