COMPUTER ORGANIZATION

AND

ASSEMBLY LANGUAGES

FINAL PROJECT

Qhasm

B95502049 王子衿

2008.01/20

WHAT?

What is Qhasm?

Assembly：

1 Programmer 自行選擇 instructions，code scheduling，以及 memory 與 Register 之操作。←在某些狀況下這是我們所要的。

2 在不同的 Architecture 下，有不同的 instruction set。

3 在不同的 Architecture 下，syntax 也不同。

↑這兩點使得 Assembly 不 portable，且對每一種 Architecture 都須重新習慣。

4 Programmer 必須自己掌握各個 variable 在哪個 Register 或是 memory 的某個區塊。←Always makes me confused…

C/C++ (High Level Language)：

1 使用之 instructions、scheduling，以及 memory 與 Register 之操作，由 compiler 執行。←Programmer 可對此方面的 optimize 有限（但仍可靠 loop unrolling 之類的方式達到一些加速）。

2 對於 instruction set 使用統一的 syntax。

Qhasm：

與 ASM 相同的部分：

1 Programmer 自行選擇 instructions，code scheduling，以及 memory 與 Register 之操作。

2 在不同的 Architecture 下，有不同的 instruction set。

與 C/C++ (High Level Language) 相同的部分：

3 使用統一的 syntax。

特殊：

4 可指定 register variables。

5 自動指定 stack space 給 stack variables。

Qhasm code and Resulting code

Qhasm(gaudry-todouble.q)	Resulting (gaudry-todouble.s)
stack32 out_stack	# stack32 out_stack
stack32 u_stack	# stack32 u_stack
int32 eax	# int32 eax
int32 (…)	(…)
int32 ebp	
int32 out	# enter gaudry_pm_todouble
int32 u	.text
	.p2align 5
int32 word0	.globl _gaudry_pm_todouble
int32 (…)	.globl gaudry_pm_todouble
int32 word4	_gaudry_pm_todouble:
float80 out0	gaudry_pm_todouble:
float80 (…)	mov %esp,%eax
float80 out4	and $31,%eax
float80 carry0	add $64,%eax
float80 (…)	sub %eax,%esp
float80 carry4	# input out_stack
stack64 d0	# input u_stack
stack64 (…)	# caller eax
stack64 d4	# caller (…)
	# caller ebp
enter gaudry_pm_todouble	
input out_stack	# u = u_stack
input u_stack	# movl <u_stack=stack32#-2,>u=int32#2
caller eax	# movl <u_stack=8(%esp,%eax),>u=%ecx
caller (…)	movl 8(%esp,%eax),%ecx
caller ebp	
	# word0 = *(uint32 *) (u + 0)
u = u_stack	# movl 0(<u=int32#2),>word0=int32#3
	# movl 0(<u=%ecx),>word0=%edx
word0 = *(uint32 *) (u + 0)	movl 0(%ecx),%edx
d0 top = 0x43300000	(…)
inplace d0 bottom = word0	# leave
(…)	add %eax,%esp
leave	ret

Qhasm 特性：

1 Qhasm 的內部不可再有 Function Calls。

2 每一行 Qhasm code 直接對應至 Resulting code 中的一個指令。

3 當某個 Register 爲 Caller-used 需要 callee-saved，作 caller Register 宣告，如此在自動存放變數時即不會使用該 Register，而不更動其值，亦不需另外儲存及讀取以保存資料。

4 目前已支援 MMX、SSE、SSE2、SSE3 等指令。

（詳細需查網址 http://cryptojedi.org/programming/qhasm-doc/）

WHY?

Why using qhasm?

Daniel J. Bernstein

Peter Schwabe worked with D.J. Bernstein

AES implementation for UltraSPARC	
gcc	25.08 cycle/byte
Sun C compiler	20.75 cycle/byte
Qhasm	15.98 cycle/byte

About 22.99% faster!!

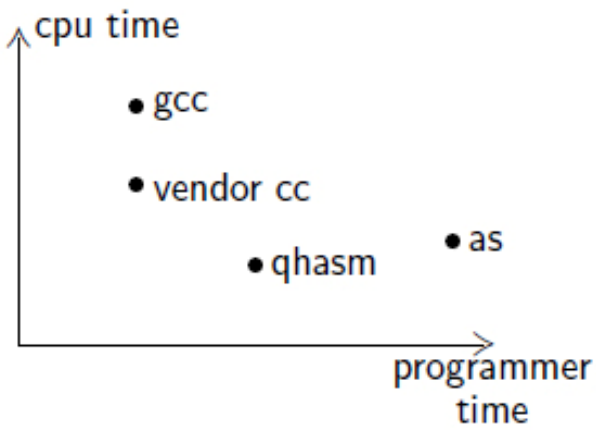
Peter Schwabe

其他平台	
UltraSPARC III	12.08 cycle/byte
PowerPC G4 7410	14.57 cycle/byte
Pentium 4 f12	14.15 cycle/byte
Core 2	10.57 cycle/byte
Athlon64	10.43 cycle/byte

All these implementations improve upon previously fastest code

2005.02.21，D. J. Bernstein 於 Fast Software Encryption 的 lecture 中

User cooperates with qhasm tools to easily achieve excellent register allocation, excellent instruction scheduling, automated range verication, etc.



Qhasm 優點簡述：

由前圖可見，Qhasm 的目標是節省大量的 coding time 並達到更高的效能；而主要使用的時機在於對程式中會反覆執行的 Function 加速。效能的部分在 AES implementation 中已可見其實際成果。而藉由 register variables 的使用，Qhasm 內建的 register allocator 會自動作適合的安排，免去 Progammer 自行思考 register 使用方式。統一的 syntax 也能減少 Progammer 的困擾，達到 coding time 上的節省。並且也支援多數的指令集供 Progammer 使用，即便有 list 中沒有的指令也能簡單地加到 list 使其能運行。

BUT!

It isn't too good to be real?

如此方便能減輕 Progammer 負擔，甚至能達到比撰寫 Assembly 更高的效能，似乎過於理想？

WARNING WARNING WARNING: qhasm is in prototype form.

Qhasm still ALPHA：

點入 Qhasm 網站(http://cr.yp.to/qhasm.html)即可看見粗黑體的三個 WARNING，以及後續的一長串敘述。說明 Qhasm 本身，.q language 還不是正式確定的定案，意即若是現在使用 Qhasm 撰寫程式，在未來 code 的維護上會出現困難。另外現有的 Qhasm tools 中有不少已知的瑕疵即使用上的不方便。雖然 Website 上說明對於每個 Qhasm tool 都有 rewrite plane，但從 Website 上追朔 Qhasm 可發現大概於 2005 年起始，而最近一次 tools 的更新時間應該是在 2007 年初…。

Not scared? Okay. Here's how to download and compile the current prototypes of qhasm's instruction decoder, register allocator, x86-floating-point-stack handler, and assembler:

```
wget http://cr.yp.to/qhasm/qhasm-20070207.tar.gz
(…)
```

↑ Qhasm 網站中一段俏皮的敘述

What's going on with Qhasm tools：

由於目前暫時沒看到其他人使用 Qhasm 撰寫程式，所以無法由 code 的維護部分做討論。而在 Qhasm tools 的部分，則是還有很大的改進空間以及未完成的部分。其中最令人困擾的可能在於其 compiler。

If you do this

int32 eax int32 ebx You can not pass!!! int32 out (…)	.q language Error Sample 1

將這份.q language 進行 compile，會很順利地"通過"，但不代表 compile 後的.s 檔案可以使用，事實上在用 GCC 與 C/C++ code 作 Link 時就會出現錯誤。

What happened?

# int32 eax # int32 ebx #You can not pass!!! ERROR #int32 out (…)	.s language Error Sample 1 ERROR message inside the .s file

雖然在 compile 的時候沒有提示任何錯誤，但很顯然的在其生成的.s 中出現了 ERROR。這使得 Progammer 在 Qhasm compile 後即刻無法確認其語法是否正確，還得進入.s 中搜尋 ERROR 訊息，而事實上不論.q 中寫了什麼，compile 都會通過。但這還不是最糟的，至少.s 中有 ERROR 訊息可詢。

What's more

# int32 pass 	
# int32 asmpass1 	
# int32 asmpass2 	
(…)
# int32 asmpass9 	
asmpass1 = *(int32 *) (pass + asmpass2)
asmpass2 = *(int32 *) (pass + asmpass3)
(…)
asmpass9 = *(int32 *) (pass + asmpass1)
	.q language
	Error Sample 2

在此宣告了 10 個 32bits register variables，並依序對其做一些運算，必須注意的是我們只有 EAX~ EDX 最多加上 EBP、 ESP、 ESI、EDI 共計 8 個 32bits Register。經過 Qhasm 的 register allocator 之後，很可能會在.s 的某行發生類似下列情況：

```
movl (%ecx),
```

很顯然這是無效的指令，但更慘的是這次連 ERROR message 都不會出現。由此可知，在 Register 數量不足，而對多數 register variables 最相近頻率的操作時，很可能會導致 Qhasm 的 register allocator 在 register 分配上的錯誤。

綜合以上，Qhasm 雖然有不少迷人的魅力，但 Qhasm tools 就不是那麼的 friendly。事實上，D. J. Bernstein 就曾在討論中說明，Qhasm 的目標爲用以加速關鍵之 functions 而非 general-purpose programming language。而在我聽的演講當中，Peter Schwabe 最後也提及 Qhasm 適合不會犯錯的 Progammer（根據其說法目前 Qhasm 的使用者僅三位）。

CONCLUSION

修習資工系的組合語言之後，才發現原來還有這麼大一塊領域是自己從來沒接觸過的，雖然在大一修習的計算機概論有寫過一些類似 TOY 的 Assembly，不過相較之下可謂皮毛。在學習組合語言的過程中，對我而言有兩大困擾，其一是每次更換架構的時候，都有種茫然感，syntax 也換，instruction 也換，loop、stack 的操作方式也有所不同，連 Compiler 也得每次安裝所對應的，常常有摸索了很久，感覺好不容易有點開竅又換了一個架構。另一個難題則是 Register 的應用，如何安排 Register 使得程式的效能能更上一層樓，往往必須思考很久，也不一定能有好成果。

Qhasm 看起來像是一計良藥，解決了很大一部分的問題，儘管目前還在 Alpha 階段，也有不少內部的問題，還不容易學習，但已有一定的成果。在 Register 數量有限時由程式算出出最佳分配是可行的，而在未來 Register 數量可能擴充，就更需要借助成是輔助優化。隨著 Compiler optimize 的能力不斷提升，我認爲像 Qhasm 這類部分由 Compiler 輔助的 language 應該是未來發展的趨勢。