Quasilyte blog

83% technical. NaN% emotional.

[POSTS BY TAGS] [REPORT AN ISSUE] [SUBSCRIBE]

Go assembly language complementary reference

20 SEPTEMBER 2017

Third flavour of x86 assembly language

This article is heavily inspired by AT&T VS Intel/Microsoft assembly syntax comparison page. It helped me very much in the past.

Many tools understand both AT&T and Intel style assembly. Go derived Plan9 toolchain, so you should learn **third** kind of assembly language. It will not accept anything else.

In this document:

- Comparison of Plan9/Go and AT&T/GNU assembly;
- Some non-obvious details that are not described anywhere else;
- Links to useful resources;

Check external resources for information that is missing here.

How to use this

Most information is presented in table-like format.
It's optimized for `ctrl+f` search over the page.

When you see {n} inside the text – look into external resources section.

Operands order

Go assembly may resemble AT&T assembly, but there are subtle differences. Operands order is one of them, but only for **some** instructions.

Below you can find a complete list of instructions that have operand order that differs from AT&T.

This list is generated from x86.csv{1} (generated by x86spec).

```
Mnemonic | Go operands              | AT&T operands
================================================
BOUNDW   | m16&16, r16              | r16, m16&16
BOUNDL   | m32&32, r32              | r32, m32&32
CMPB     | AL, imm8                 | imm8, AL
CMPW     | AX, imm16                | imm16, AX
CMPL     | EAX, imm32               | imm32, EAX
CMPQ     | RAX, imm32               | imm32, RAX
CMPW     | r/m16, imm16             | imm16, r/m16
CMPW     | r/m16, imm8              | imm8, r/m16
CMPW     | r/m16, r16               | r16, r/m16
CMPL     | r/m32, imm32             | imm32, r/m32
CMPL     | r/m32, imm8              | imm8, r/m32
CMPL     | r/m32, r32               | r32, r/m32
CMPQ     | r/m64, imm32             | imm32, r/m64
CMPQ     | r/m64, imm8              | imm8, r/m64
CMPQ     | r/m64, r64               | r64, r/m64
CMPB     | r/m8, imm8               | imm8, r/m8
CMPB     | r/m8, imm8               | imm8, r/m8
CMPB     | r/m8, r8                 | r8, r/m8
CMPB     | r/m8, r8                 | r8, r/m8

CMPW     | r16, r/m16               | r/m16, r16
```

```
CMPL     | r32, r/m32             | r/m32, r32
CMPQ     | r64, r/m64             | r/m64, r64
CMPB     | r8, r/m8               | r/m8, r8
CMPB     | r8, r/m8               | r/m8, r8
CMPPD    | imm8, xmm1, xmm2/m128 | imm8, xmm2/m128, xmm1
CMPPS    | imm8, xmm1, xmm2/m128 | imm8, xmm2/m128, xmm1
CMPSD    | imm8, xmm1, xmm2/m64  | imm8, xmm2/m64, xmm1
CMPSS    | imm8, xmm1, xmm2/m32  | imm8, xmm2/m32, xmm1
ENTER    | 0, imm16               | imm16, 0
ENTER    | 1, imm16               | imm16, 1

ENTERW/ENTERL/ENTERQ | imm8b, imm16 | imm16, imm8b
```

This table is useful if you can not interpret this quote from official manual{2}:

> *"The order of operands is from left to right in dataflow order, just as on the 68020 but not as in MIPS documentation."*

Register operand rules

Before showing divergent instruction names, important part of official manual{2} should be quoted and explained.

> *All registers are 64 bit, but instructions access low-order 8, 16 and 32 bits as described in the processor handbook. For example, MOVL to AX puts a value in the low-order 32 bits and clears the top 32 bits to zero.*

Little snippet below will demonstrate the idea:

```
Go        | AT&T
================
BSWAPW AX | BSWAP %ax
BSWAPL AX | BSWAP %eax
BSWAPQ AX | BSWAP %rax
```

When instruction supports multiple operand sizes, Plan9 assembly uses pseudo instructions like `BSWAP{W,L,Q}` above, that specify operand size.

The next two sections should be easier to grasp with this knowledge.

Mnemonics

Many mnemonics are different from AT&T.

With the exception of `RETFW/RETFL/RETFQ` (far ret), instruction names differ in suffixes only.

AVX instructions all have Intel names. That is, we have `VADDPD` rather than `VADDPL`.

Instructions that have `CC` (condition) part in them historically have different spelling in Go:

> *The GNU form is CMOV{cond}{size} but the form added for Go is CMOV{size}{cond}. So GNU's cmovleq is cmov-le-q while Go's CMOVLEQ is CMOV-L-EQ*

There is an open issue for `CMOVLEQ` instruction. This may lead to `CMOVL.EQ` syntax in future.

> *The names of conditions in all conditional instructions (J, SET) follow the conventions of the 68020 instead of those of the Intel assembler*

```
Go          | AT&T        | Operands
==================================
BSWAPW      | BSWAP       | r16op
BSWAPL      | BSWAP       | r32op
BSWAPQ      | BSWAP       | r64op
CBW         | CBTW        |
CDQ         | CLTD        |
CDQE        | CLTQ        |
CMOVWHI     | CMOVAW      | r/m16, r16
CMOVLHI     | CMOVAL      | r/m32, r32
CMOVQHI     | CMOVAQ      | r/m64, r64
CMOVWCC     | CMOVAEW     | r/m16, r16
CMOVLCC     | CMOVAEL     | r/m32, r32
CMOVQCC     | CMOVAEQ     | r/m64, r64
CMOVWCS     | CMOVBW      | r/m16, r16
CMOVLCS     | CMOVBL      | r/m32, r32
CMOVQCS     | CMOVBQ      | r/m64, r64
CMOVWLS     | CMOVBEW     | r/m16, r16
CMOVLLS     | CMOVBEL     | r/m32, r32
CMOVQLS     | CMOVBEQ     | r/m64, r64
CMOVWEQ     | CMOVEW      | r/m16, r16
CMOVLEQ     | CMOVEL      | r/m32, r32
CMOVQEQ     | CMOVEQ      | r/m64, r64
CMOVWGT     | CMOVGW      | r/m16, r16
CMOVLGT     | CMOVGL      | r/m32, r32
CMOVQGT     | CMOVGQ      | r/m64, r64
CMOVWGE     | CMOVGEW     | r/m16, r16
CMOVLGE     | CMOVGEL     | r/m32, r32
CMOVQGE     | CMOVGEQ     | r/m64, r64
```

```
CMOVWLT    | CMOVLW     | r/m16, r16
CMOVLLT    | CMOVLL     | r/m32, r32
CMOVQLT    | CMOVLQ     | r/m64, r64
CMOVWLE    | CMOVLEW    | r/m16, r16
CMOVLLE    | CMOVLEL    | r/m32, r32
CMOVQLE    | CMOVLEQ    | r/m64, r64
CMOVWNE    | CMOVNEW    | r/m16, r16
CMOVLNE    | CMOVNEL    | r/m32, r32
CMOVQNE    | CMOVNEQ    | r/m64, r64
CMOVWOC    | CMOVNOW    | r/m16, r16
CMOVLOC    | CMOVNOL    | r/m32, r32
CMOVQOC    | CMOVNOQ    | r/m64, r64
CMOVWPC    | CMOVNPW    | r/m16, r16
CMOVLPC    | CMOVNPL    | r/m32, r32
CMOVQPC    | CMOVNPQ    | r/m64, r64
CMOVWPL    | CMOVNSW    | r/m16, r16
CMOVLPL    | CMOVNSL    | r/m32, r32
CMOVQPL    | CMOVNSQ    | r/m64, r64
CMOVWOS    | CMOVOW     | r/m16, r16
CMOVLOS    | CMOVOL     | r/m32, r32
CMOVQOS    | CMOVOQ     | r/m64, r64
CMOVWPS    | CMOVPW     | r/m16, r16
CMOVLPS    | CMOVPL     | r/m32, r32
CMOVQPS    | CMOVPQ     | r/m64, r64
CMOVWMI    | CMOVSW     | r/m16, r16
CMOVLMI    | CMOVSL     | r/m32, r32
CMOVQMI    | CMOVSQ     | r/m64, r64
CQO        | CQTO       |
CVTPL2PD   | CVTDQ2PD   | xmm2/m64, xmm1
CVTPL2PS   | CVTDQ2PS   | xmm2/m128, xmm1
CVTPD2PL   | CVTPD2DQ   | xmm2/m128, xmm1
CVTPS2PL   | CVTPS2DQ   | xmm2/m128, xmm1
CVTSD2SL   | CVTSD2SI   | xmm2/m64, r32
CVTSD2SL   | CVTSD2SIQ  | xmm2/m64, r64
CVTSL2SD   | CVTSI2SDL  | r/m32, xmm1
CVTSQ2SD   | CVTSI2SDQ  | r/m64, xmm1
CVTSL2SS   | CVTSI2SSL  | r/m32, xmm1
CVTSQ2SS   | CVTSI2SSQ  | r/m64, xmm1
CVTSS2SL   | CVTSS2SI   | xmm2/m32, r32
CVTSS2SL   | CVTSS2SIQ  | xmm2/m32, r64
CVTTPD2PL  | CVTTPD2DQ  | xmm2/m128, xmm1
CVTTPS2PL  | CVTTPS2DQ  | xmm2/m128, xmm1
CVTTSD2SL  | CVTTSD2SI  | xmm2/m64, r32
CVTTSD2SL  | CVTTSD2SIQ | xmm2/m64, r64
CVTTSS2SL  | CVTTSS2SI  | xmm2/m32, r32
CVTTSS2SL  | CVTTSS2SIQ | xmm2/m32, r64
CWD        | CWTD       |
CWDE       | CWTL       |
FADDD      | FADD       | ST(i), ST(0)
FADDD      | FADD       | ST(0), ST(i)
FADDD      | FADDS      | m32fp
FADDD      | FADDL      | m64fp
FADDDP     | FADDP      |
FADDDP     | FADDP      | ST(0), ST(i)
FCOMD      | FCOM       |
FCOMD      | FCOM       | ST(i)
FCOMD      | FCOMS      | m32fp
FCOMD      | FCOML      | m64fp
```

```
FCOMFP     | FCOMPS     | m32fp
FDIVD      | FDIV       | ST(i), ST(0)
FDIVD      | FDIVR      | ST(0), ST(i)
FDIVD      | FDIVS      | m32fp
FDIVD      | FDIVL      | m64fp
FDIVD      | FDIV       | ST(0), ST(i)
FDIVFR     | FDIVRS     | m32fp
LOOPEQ     | LOOPE      | rel8
MASKMOVOU  | MASKMOVDQU | xmm2, xmm1
MOVQ       | MOVABSQ    | moffs64, RAX
MOVQ       | MOVABSQ    | RAX, moffs64
MOVQ       | MOVABSQ    | imm64, r64op
MOVQ       | MOVDQ2Q    | xmm2, mm1
MOVO       | MOVDQA     | xmm2/m128, xmm1
MOVO       | MOVDQA     | xmm1, xmm2/m128
MOVOU      | MOVDQU     | xmm2/m128, xmm1
MOVOU      | MOVDQU     | xmm1, xmm2/m128
MOVNTO     | MOVNTDQ    | xmm1, m128
MOVQOZX    | MOVQ2DQ    | mm2, xmm1
MOVBWSX    | MOVSBW     | r/m8, r16
MOVWLSX    | MOVSWL     | r/m16, r32
MOVBLSX    | MOVSBL     | r/m8, r32
MOVWQSX    | MOVSWQ     | r/m16, r64
MOVBQSX    | MOVSBQ     | r/m8, r64
MOVWQSX    | MOVSXDW    | r/m32, r16
MOVLQSX    | MOVSXDL    | r/m32, r32
MOVLQSX    | MOVSLQ     | r/m32, r64
MOVBWZX    | MOVZBW     | r/m8, r16
MOVWLZX    | MOVZWL     | r/m16, r32
MOVBLZX    | MOVZBL     | r/m8, r32
MOVWQZX    | MOVZWQ     | r/m16, r64
MOVBQZX    | MOVZBQ     | r/m8, r64
PACKSSLW   | PACKSSDW   | mm2/m64, mm1
PACKSSLW   | PACKSSDW   | xmm2/m128, xmm1
PADDL      | PADDD      | mm2/m64, mm1
PADDL      | PADDD      | xmm2/m128, xmm1
PCMPEQL    | PCMPEQD    | mm2/m64, mm1
PCMPEQL    | PCMPEQD    | xmm2/m128, xmm1
PCMPGTL    | PCMPGTD    | mm2/m64, mm1
PCMPGTL    | PCMPGTD    | xmm2/m128, xmm1
PMADDWL    | PMADDWD    | mm2/m64, mm1
PMADDWL    | PMADDWD    | xmm2/m128, xmm1
PMULULQ    | PMULUDQ    | mm2/m64, mm1
PMULULQ    | PMULUDQ    | xmm2/m128, xmm1
PSLLL      | PSLLD      | mm2/m64, mm1
PSLLL      | PSLLD      | imm8, mm2
PSLLL      | PSLLD      | xmm2/m128, xmm1
PSLLL      | PSLLD      | imm8, xmm2
PSLLO      | PSLLDQ     | imm8, xmm2
PSRAL      | PSRAD      | mm2/m64, mm1
PSRAL      | PSRAD      | imm8, mm2
PSRAL      | PSRAD      | xmm2/m128, xmm1
PSRAL      | PSRAD      | imm8, xmm2
PSRLL      | PSRLD      | mm2/m64, mm1
PSRLL      | PSRLD      | imm8, mm2
PSRLL      | PSRLD      | xmm2/m128, xmm1
PSRLL      | PSRLD      | imm8, xmm2
PSRLO      | PSRLDQ     | imm8, xmm2
```

```
PSUBL      | PSUBD      | mm2/m64, mm1
PSUBL      | PSUBD      | xmm2/m128, xmm1
PUNPCKHLQ  | PUNPCKHDQ  | mm2/m64, mm1
PUNPCKHLQ  | PUNPCKHDQ  | xmm2/m128, xmm1
PUNPCKHWL  | PUNPCKHWD  | mm2/m64, mm1
PUNPCKHWL  | PUNPCKHWD  | xmm2/m128, xmm1
PUNPCKLLQ  | PUNPCKLDQ  | mm2/m32, mm1
PUNPCKLLQ  | PUNPCKLDQ  | xmm2/m128, xmm1
PUNPCKLWL  | PUNPCKLWD  | mm2/m32, mm1
PUNPCKLWL  | PUNPCKLWD  | xmm2/m128, xmm1
SETHI      | SETA       | r/m8
SETCC      | SETAE      | r/m8
SETCS      | SETB       | r/m8
SETLS      | SETBE      | r/m8
SETCS      | SETC       | r/m8
SETEQ      | SETE       | r/m8
SETGT      | SETG       | r/m8
SETLT      | SETL       | r/m8
SETLS      | SETNA      | r/m8
SETCS      | SETNAE     | r/m8
SETCC      | SETNB      | r/m8
SETHI      | SETNBE     | r/m8
SETCC      | SETNC      | r/m8
SETLE      | SETNG      | r/m8
SETLT      | SETNGE     | r/m8
SETGE      | SETNL      | r/m8
SETGT      | SETNLE     | r/m8
SETOC      | SETNO      | r/m8
SETPC      | SETNP      | r/m8
SETPL      | SETNS      | r/m8
SETNE      | SETNZ      | r/m8
SETOS      | SETO       | r/m8
SETPS      | SETP       | r/m8
SETPS      | SETPE      | r/m8
SETPC      | SETPO      | r/m8
SETMI      | SETS       | r/m8
SETEQ      | SETZ       | r/m8
SHLW       | SHLDW      | CL, r16, r/m16
SHLW       | SHLDW      | imm8, r16, r/m16
SHLL       | SHLDL      | CL, r32, r/m32
SHLL       | SHLDL      | imm8, r32, r/m32
SHLQ       | SHLDQ      | CL, r64, r/m64
SHLQ       | SHLDQ      | imm8, r64, r/m64
SHRW       | SHRDW      | CL, r16, r/m16
SHRW       | SHRDW      | imm8, r16, r/m16
SHRL       | SHRDL      | CL, r32, r/m32
SHRL       | SHRDL      | imm8, r32, r/m32
SHRQ       | SHRDQ      | CL, r64, r/m64
SHRQ       | SHRDQ      | imm8, r64, r/m64

SYSRET             | SYSRETW/SYSRETL/SYSRETL |
RETFW/RETFL/RETFQ  | LRETW/LRETL/LRETL       | imm16u
```

For some instructions there are aliases available. They are undocumented.

To get up-to-date list of aliases, inspect arch.go `archX86` function.

```
/* file "arch.go" */

// alias "MASKMOVDQU"->"MASKMOVOU"
instructions["MASKMOVDQU"] = x86.AMASKMOVOU
// alias "MOVD"->"MOVQ"
instructions["MOVD"] = x86.AMOVQ
// ... more aliases
```

Go 1.9 x86 alias list file is provided for convenience.

As of June 2018, x86.csv mentioned above does not provide correct syntax for some Go instructions.

Known issues

List below is a hand-written collection of known issues:

- `FIADD m16int` is `FADDW`, but disassembled as `FIADD`
- `FIADD m32int` is `FADDL`, but disassembled as `FIADD`
- Issue 20111: `CMOVLGE` and `CMOVQGE` as `CMOVGE`
- Issue 23386: `FSAVE` assembled into `FNSAVE`; Real `FSAVE` is not implemented

Register names

Tables below map AT&T <=> Go register names.

```
%ah <=> AH
%bh <=> BH
%ch <=> CH
%dh <=> DH

%al <=> AL
%bl <=> BL
%cl <=> CL
%dl <=> DL

%sil       <=> SIB
%dil       <=> DIB
%bpl       <=> BPB
%spl       <=> SPB
%r8b-%r15b <=> R8B-R15B
```

```
%xmm0-%xmm31 <=> X0-X31
%ymm0-%ymm31 <=> Y0-Y31
%zmm0-%zmm31 <=> Z0-Z31

%k0-%k7 <=> K0-K7

%st0-%st7 <=> F0-F7
%mm0-%mm7 <=> M0-M7

%cs, %ss, %ds, %es, %ds, %gs <=> CS, SS, DS, ES, DS, GS

%cr0-%cr7 <=> CR0-CR7
%db0-%db7 <=> DR0-DR7
%tr0-%tr7 <=> TR0-TR7

;; TODO: GDTR, IDTR, LDTR, MSG, TASK
;; TODO: TLS
```

Notes:

- High 16 `xmm` / `ymm` , all `zmm` and `k` registers are enabled in AVX512{4}
- The exact count of defined `CR` , `DR` and `TR` register may vary (up to 16);

In **64-bit** mode you can not use instructions with `Q` suffix. Effectively, this means that you can not treat registers like `AX` as 64-bit wide. Read about `GOARCH`{3} for more information.

```
32-bit    | 64-bit only
=======================
%eax      | %rax     <=> AX
%ecx      | %rcx     <=> CX
%edx      | %rdc     <=> DX
%ebx      | %rbx     <=> BX
%esp      | %rsp     <=> SP
%sbp      | %rbp     <=> BP
%esi      | %rsi     <=> SI
%edi      | %rdi     <=> DI
%r8d-r15d | %r8-%r15 <=> R8-R15
```

Syntax oddities

1. The `REG0:REG1` syntax expands into 2 register operands, in reverse order.
   So, `MOVL AX:DX` becomes `MOVL DX, AX` .
   The latter syntax is recommended.

2. It's almost always OK to add `*` in front of register.
   `MOVL *DX, *AX` is the same as `MOVL DX, AX`.

Forbidden constructions

1. You can not use SIB addressing without explicit S-scale.
   Issue 21860 mentions this restriction.

2. Immediate operands are typed. They can be either signed or unsigned. You can not use negative constant arguments for instruction if it has "unsigned" immediate restriction.
   Proposal 21528 has discussion on related subject.

3. Pseudo register `IZ` (`%riz` , `%eiz`) does not exist.
   Issue 18792, although indirectly, confirms that.

External resources

This is "further reading" section.

- [1] My x86.csv v0.2 copy.
- Public, but "unstable" x86.csv
- [2] Manual for the Plan 9 assembler by Rob Pike.
- Quick Guide to Go's Assembler from go docs.
- [3] Go build docs: environment variables.
- [4] AVX512 design.

[GO] [ASM] [PLAN9] [REFERENCE]

BACK TO TOP SHARE

Iskander Sharipov
Lisper that lost in a gophers land