REFERENCE MATERIAL FOR COS217 FINAL EXAM

A Subset of x86-64 Assembly Language

Syntax	Semantics	Description
`mov{q,l,w,b} srcIRM, destRM`	`dest = src;`	**Move**. Copy *src* to *dest.*Flags affected: None.
`push{q,w} srcIRM`	`reg[RSP] = reg[RSP] - {8,2};` `mem[reg[RSP]] = src;`	**Push**. Push *src* onto the stack. Flags affected: None.
`pop{q,w} destRM`	`dest = mem[reg[RSP]];` `reg[ESP] = reg[RSP] + {8,2};`	**Pop**. Pop from the stack into *dest.* Flags affected: None.
`lea{q,l,w} srcM, destR`	`dest = &src;`	**Load Effective Address**. Assign the address of *src* to *dest*. That is, determine the address denoted by *src*, but don't fetch data from that address; instead use the address itself. Flags affected: None.
`add{q,l,w,b} srcIRM, destRM`	`dest = dest + src;`	**Add**. Add *src* to *dest.* Flags affected: O, S, Z, A, C, P.
`add{q,l,w,b} srcIRM, destRM`	`dest = dest + src;`	**Add**. Add *src* to *dest.* Flags affected: O, S, Z, A, C, P.
`imul{q,l,w} srcIRM, destR`	`dest = dest * src;`	**Multiply**. Multiply *dest* by *src*. Flags affected: O, S, Z, A, C, P.
`imulq srcRM`	`reg[RDX:RAX] = reg[RAX]*src;`	**Signed Multiply**. Multiply the contents of register RAX by *src*, and store the product in registers RDX:RAX. Flags affected: O, S, Z, A, C, P.
`imull srcRM`	`reg[EDX:EAX] = reg[EAX]*src;`	**Signed Multiply**. Multiply the contents of register EAX by *src*, and store the product in registers EDX:EAX. Flags affected: O, S, Z, A, C, P.
`idivq srcRM`	`reg[RAX] = reg[RDX:RAX]/src;` `reg[RDX] = reg[RDX:RAX]%src;`	**Signed Divide**. Divide the contents of registers RDX:RAX by *src*, and store the quotient in register RAX and the remainder in register RDX. Flags affected: O, S, Z, A, C, P.
`idivl srcRM`	`reg[EAX] = reg[EDX:EAX]/src;` `reg[EDX] = reg[EDX:EAX]%src;`	**Signed Divide**. Divide the contents of registers EDX:EAX by *src*, and store the quotient in register EAX and the remainder in register EDX. Flags affected: O, S, Z, A, C, P.
`mulq srcRM`	`reg[RDX:RAX] = reg[RAX]*src;`	**Unsigned Multiply**. Multiply the contents of register RAX by *src*, and store the product in registers RDX:RAX. Flags affected: O, S, Z, A, C, P.
`mull srcRM`	`reg[EDX:EAX] = reg[EAX]*src;`	**Unsigned Multiply**. Multiply the contents of register EAX by *src*, and store the product in registers EDX:EAX. Flags affected: O, S, Z, A, C, P.
`sal{q,l,w,b} srcIR, destRM`	`dest = dest << src;`	**Shift Arithmetic Left**. Shift *dest* to the left *src* bits, filling with zeros. If *src* is a register, then it must be the CL register. Flags affected: O, S, Z, A, C, P.
`sar{q,l,w,b} srcIR, destRM`	`dest = dest >> src;`	**Shift Arithmetic Right**. Shift *dest* to the right *src* bits, sign extending the number. If *src* is a register, then it must be the CL register. Flags affected: O, S, Z, A, C, P.
`cmp{q,l,w,b} srcIRM, destRM`	`reg[EFLAGS] =` `dest comparedWith src;`	**Compare**. Compute *dest* - *src* and set flags in the EFLAGS register based upon the result. Flags affected: O, S, Z, A, C, P.
`test{q,l,w,b} srcIRM, destRM`	`reg[EFLAGS] = dest & src;`	**Test**. Compute *dest* & *src* and set flags in the EFLAGS register based upon the result. Flags affected: S, Z, P (O and C set to 0).

`jmp label`	`reg[RIP] = label;`	**Jump**. Jump to *label*. Flags affected: None.
`jmp *srcRM`	`reg[RIP] = reg[src];`	**Jump indirect.** Jump to the address in *src*. Flags affected: None.

`j{e,ne,`
`l,le,g,ge,`
`b,be,a,ae} label`
	`if (reg[EFLAGS] appropriate)`
	`reg[RIP] = label;`
		Conditional Jump. Jump to *label* iff the flags in the EFLAGS register indicate a(n) equal to, unequal to, less than, less than or equal to, greater than, greater than or equal to, below, below or equal to, above, or above or equal to (respectively) relationship between the most recently compared numbers. The l, le, g, and ge forms are used after comparing signed numbers; the b, be, a, and ae forms are used after comparing unsigned numbers. Flags affected: None.

`call label`	`reg[RSP] = reg[RSP] - 8;`
	`mem[reg[RSP]] = reg[RIP];`
	`reg[RIP] = label;`
		Call. Call the function that begins at *label.* Flags affected: None.

`call *srcRM`	`reg[RSP] = reg[RSP] - 8;`
	`mem[reg[RSP]] = reg[RIP];`
	`reg[RIP] = reg[src];`
		Call indirect. Call the function whose address is in *src*. Flags affected: None.

`Ret`	`reg[RIP] = mem[reg[RSP]];` `reg[RSP] = reg[RSP] + 8;`	**Return**. Return from the current function. Flags affected: None.

Syntax	Description
`label:`	Record the fact that *label* marks the current location within the current section.
`.section ".sectionname"`	Make the *sectionname* section the current section.
`.skip n`	Skip *n* bytes of memory in the current section.
`.long longvalue1, longvalue2, ...`	Allocate four bytes of memory containing *longvalue1*, four bytes of memory containing *longvalue2*, ... in the current section.
`.quad quadvalue1, quadvalue2, ...`	Allocate eight bytes of memory containing *quadvalue1*, eight bytes of memory containing *quadvalue2*, ... in the current section.
`.globl label1, label2, ...`	Mark *label1*, *label2*, ... so they are accessible by code generated from other source code files.
`.equ name, expr`	Define *name* as a symbolic alias for *expr.*
`.type label,@function`	Mark *label* so the linker knows that it denotes the beginning of a function.

General purpose registers

Arguments: rdi, rsi, rdx, rcx, r8, r9
Caller-saved: Arguments + rax, r10, r11
Callee-saved: rbx, rbp, r12, r13, r14, r15,
Stack pointer: rsp

Operands

Type	From	Operand Value	Name
Immediate	$Imm	Imm	Immediate
Register	%r	R[%r]	Register
Memory	Imm	M[Imm]	Absolute
Memory	(%r)	M[R[%r]]	Indirect
Memory	d(%r)	M[d+R[%r]]	Base+Displacement
Memory	d(,%r,n)	M[d+R[%r]*n]	Scaled Indexed
Memory	d(%b,%r,n)	M[d+R[%b]+R[%r]*n]	Scaled Indexed with base