Y86-64 Instructions Encoding

Byte	0		1		2	3	4	5	6	7	8	9
halt	0	0										
nop	1	0										
rrmovq **rA**, **rB**	2	0	**rA**	**rB**								
cmovXX **rA**, **rB**	2	fn	**rA**	**rB**								
irmovq **V**, **rB**	3	0	F	**rB**	**V**							
rmmovq **rA**, **D**(**rB**)	4	0	**rA**	**rB**	**D**							
mrmovq **D**(**rB**), **rA**	5	0	**rA**	**rB**	**D**							
OPq **rA**, **rB**	6	fn	**rA**	**rB**								
jXX **Dest**	7	fn	**Dest**									
call **Dest**	8	0	**Dest**									
ret	9	0										
pushq **rA**	A	0	**rA**	F								
popq **rA**	B	0	**rA**	F								

Instruction	Semantics	Example
rrmovq %rs, %rd	r[rd] ← r[rs]	rrmovq %rax, %rbx
cmovXX %rs, %rd	r[rd] ← r[rs] if last ALU XX 0 (XX is le/l/e/ne/ge/g)	cmovle %rax, %rbx
irmovq $i, %rd	r[rd] ← i	irmovq $100, %rax
rmmovq %rs, D(%rd)	m[D + r[rd]] ← r[rs]	rmmovq %rax, 100(%rbx)
mrmovq D(%rs), %rd	r[rd] ← m[D + r[rs]]	mrmovq 100(%rbx), %rax
OPq %rs, %rd	r[rd] ← r[rd] OP r[rs]	addq %rax, %rbx
jmp D	goto D	jmp foo
jXX D	goto D if last ALU result XX 0 (XX is le/l/e/ne/ge/g)	jle foo
call D	pushq PC; jmp D	call foo
ret	popq PC	ret
pushq %rs	m[r[rsp] - 8] ← r[rs]; r[rsp] = r[rsp] - 8	pushq %rax
popq %rd	r[rd] ← m[r[rsp]]; r[rsp] = r[rsp] + 8	popq %rax

Hexadecimal conversions

Hex	Bin	Hex	Bin
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

ifun values

ifun	OPq	jXX/cmovXX
0	add	no condition
1	sub	le
2	and	l
3	xor	e
4	mul	ne
5	div	ge
6	mod	g

Register Names

#	Name	#	Name
0	%rax	8	%r8
1	%rcx	9	%r9
2	%rdx	A	%r10
3	%rbx	B	%r11
4	%rsp	C	%r12
5	%rbp	D	%r13
6	%rsi	E	%r14
7	%rdi	F	NONE

Function argument registers: %rdi %rsi %rdx %rcx %r8 %r9 Function return register: %rax