

## Some Details About Y86-64 & x86-64

RF: Program registers

0 %rax	4 %rsp	%r8	%r12
1 %rcx	5 %rbp	%r9	%r13
2 %rdx	6 %rsi	%r10	%r14
3 %rbx	7 %rdi	%r11	

CC: Condition codes

ZF SF OF

PC

Main Memory:  
• byte addressable array

g. babic

Y86-64 & x86-64 Details

1

## Y86-64 Instruction Formats

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

Y86-64 & x86-64 Details

2

## Y86-64 Arithmetic and Logical Instructions

Add											
addq rA, rB		6	0	rA	rB						
											$[rA] + [rB] \rightarrow rB; [PC] + 2 \rightarrow PC$
Subtract											
subq rA, rB		6	1	rA	rB						
											$[rB] - [rA] \rightarrow rB; [PC] + 2 \rightarrow PC$
And											
andq rA, rB		6	2	rA	rB						
											$[rA] \& [rB] \rightarrow rB; [PC] + 2 \rightarrow PC$
Exclusive-Or											
xorq rA, rB		6	3	rA	rB						
											$[rA] \wedge [rB] \rightarrow rB; [PC] + 2 \rightarrow PC$

- As side effect, each of these instructions sets:
  - ZF flag to 1 if the result is zero, to 0 otherwise,
  - SF flag to the value of the most significant bit of the result,
  - OF flag to 1 if there is 2's complement overflow, 0 otherwise.

g. babic

Y86-64 & x86-64 Details

3

## Y86-64 Reg-to-Reg Move Instructions

Move Unconditionally											
rmmovq rA, rB		2	0	rA	rB						
											$[rA] \rightarrow rB$
Move When Less or Equal											
cmovle rA, rB		2	1	rA	rB						
Move When Less											
cmovl rA, rB		2	2	rA	rB						
Move When Equal											
cmove rA, rB		2	3	rA	rB						
Move When Not Equal											
cmovne rA, rB		2	4	rA	rB						
Move When Greater or Equal											
cmovge rA, rB		2	5	rA	rB						
Move When Greater											
cmovg rA, rB		2	6	rA	rB						

- $rmmovl$  instruction copies value from source register rA to destination register rB, i.e.  $[rA] \rightarrow rB$

- All other reg-to-reg move instructions conditionally copy value from source register rA to destination register rB, based on values of condition codes.

- All these instructions also increment PC by 2, i.e.  $[PC] + 2 \rightarrow PC$

g. babic

Y86-64 & x86-64 Details

4

## Y86-64 Immediate & Memory Move Instructions

Move Immediate to Register											
irmovq V, rB		3	0	0xF	rB						V
											$V \rightarrow rB$
Move Register to Memory											
rmmovq rA, D(rB)		4	0	rA	rB						D
											$[rA] \rightarrow \text{Memory}[[rB] + D]$
Move Memory to Register											
rmmovq D(rB), rA		5	0	rA	rB						D
											$\text{Memory}[[rB] + D] \rightarrow rA$

- Memory address calculation for move instructions:  
 $\text{memory address} = [rB] + D$
- Note: register id 15 (0xF) indicates "no register".
- All these instructions increment PC by 10, i.e.  $[PC] + 10 \rightarrow PC$

g. babic

Y86-64 & x86-64 Details

5

## Y86-64 Jump Instructions

Jump Unconditionally											
jmp Destination		7	0								Destination
											$\text{jmp instruction copies value from its Destination field to PC, i.e. } [Destination] \rightarrow PC$
Jump When Less or Equal											
jle Destination		7	1								Destination
Jump When Less											
j1 Destination		7	2								Destination
Jump When Equal											
jle Destination		7	3								Destination
Jump When Not Equal											
jne Destination		7	4								Destination
Jump When Greater or Equal											
jge Destination		7	5								Destination
Jump When Greater											
jg Destination		7	6								Destination

- $\text{jmp}$  instruction copies value from its Destination field to PC, i.e.  $[Destination] \rightarrow PC$

- Each other jump instruction copies a value from its Destination field to PC if jump condition is satisfied, otherwise PC value is incremented by 9 (the length of jump instruction)

g. babic

Y86-64 & x86-64 Details

6

## Y86-64 Jump Instructions (cont.)

- **jle** instruction (jump when Less or Equal)
  - jump condition:  $(SF \wedge OF) \vee ZF$
- **jl** instruction (jump when Less)
  - jump condition:  $SF \wedge OF$
- **je** instruction (jump when Equal)
  - jump condition:  $ZF$
- **jne** instruction (jump when Not Equal)
  - jump condition:  $\sim ZF$
- **jge** instruction (jump when Greater or Equal)
  - jump condition:  $\sim(SF \wedge OF)$
- **jg** instruction (jump when Greater)
  - jump condition:  $\sim(SF \wedge OF) \ \& \ \sim ZF$
- Move conditions are the same as these jump conditions.

g. babic

Y86-64 & x86-64 Details

7

## Y86-64 Stack Effecting Instructions

- pushq rA**    **A** 0 **rA** 0xF

**popq rA**    **B** 0 **rA** 0xF
- Decrement  $\%rsp$  by 8
  - Store word from rA to memory at  $(\%rsp)$
  - $[PC] + 2 \rightarrow PC$
  - Read word from memory at  $(\%rsp)$
  - Save in rA
  - Increment  $\%rsp$  by 8
  - $[PC] + 2 \rightarrow PC$

- call Destination** 8 0 **Destination**
- pushes value  $PC+9$  onto stack
  - $[Destination] \rightarrow PC$

- ret**    9 0

**nop**    1 0
- pops value from stack and loads it into PC
  - $[PC] + 1 \rightarrow PC$

g. babic

Y86-64 & x86-64 Details

8

## Useful x86-64 Instructions

- **cmpq** instruction is used for setting of condition codes.
  - Format: **cmpq src2, src1**; like computing  $(a - b)$  without setting destination, where  $[src2] \rightarrow b$  and  $[src1] \rightarrow a$
- **testq** instruction is also used for setting of condition codes.
  - Format: **testq src2, src1**; like computing  $a \ \& \ b$  without setting destination, where  $[src2] \rightarrow b$  and  $[src1] \rightarrow a$
  - OF and CF set to 0.
- **leaq** instruction is a variant of the **movl** instruction; used to compactly describe common arithmetic operations
  - Format: **leaq c(%ri,%rj,s), %rk**
  - effect:  $c + [\%ri] + [\%rj] * s \rightarrow \%rk$
  - c any integer, %ri, %rj, %rk any register, s=1,2,4 or 8; c1, %ri, %rj & s optional.

g. babic

Y86-64 & x86-64 Details

9

## Examples of x86-64 Instructions

Format	Computation	
<b>addq Src, Dest</b>	$Dest = Dest + Src$	
<b>subq Src, Dest</b>	$Dest = Dest - Src$	
<b>imulq Src, Dest</b>	$Dest = Dest * Src$	
<b>salq Src, Dest</b>	$Dest = Dest \ll Src$	Also called <b>shlq</b>
<b>sarq Src, Dest</b>	$Dest = Dest \gg Src$	Arithmetic
<b>shrq Src, Dest</b>	$Dest = Dest \gg Src$	Logical
<b>xorq Src, Dest</b>	$Dest = Dest \wedge Src$	
<b>andq Src, Dest</b>	$Dest = Dest \ \& \ Src$	
<b>orq Src, Dest</b>	$Dest = Dest \vee Src$	
<b>incq Dest</b>	$Dest = Dest + 1$	
<b>decq Dest</b>	$Dest = Dest - 1$	
<b>negq Dest</b>	$Dest = -Dest$	
<b>notq Dest</b>	$Dest = \sim Dest$	

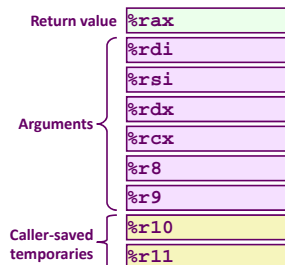
g. babic

Y86-64 & x86-64 Details

10

## Linux & Windows X86-64 Register Usage

- **%rax**  
return value, also caller-saved (if needed), since can be modified by called function
- **%rdi, %rsi, %rdx ..., %r9**  
arguments (1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, ..., 6<sup>th</sup>) also caller-saved (if needed), since can be modified by called function
- **%r10, %r11**  
caller-saved (if needed), since can be modified by called function



Y86-64 & x86-64 Details

11

## Linux & Windows X86-64 Register Usage (cont.)

- **%rbx, %r12, %r13, %r14, %rbp**  
If using them, called function must save & restore, since caller expect them to be unchanged
- **%rsp**  
Special form of called function save. Should be restored to original value upon exit from called function



g. babic

Y86-64 & x86-64 Details

12