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	

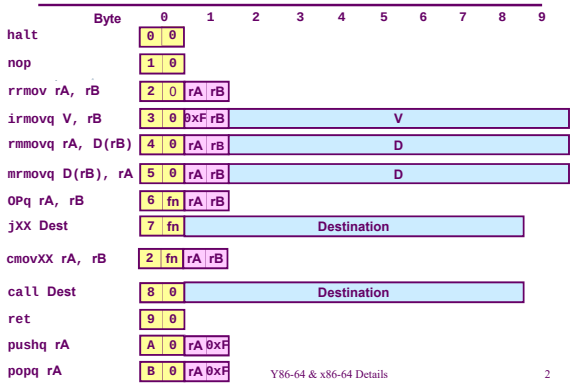
CC: Condition codes

ZF	SF	OF

PC

Main Memory:
- byte addressable array

Y86-64 Instruction Formats



Y86-64 Arithmetic and Logical Instructions

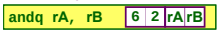
Add

[rA] + [rB] → rB; [PC]+2 →PC

Subtract

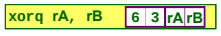
[rB] – [rA] → rB; [PC]+2 →PC

And



[rA] & [rB] → rB; [PC]+2 →PC

Exclusive-Or

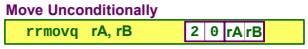


[rA] ^ [rB] → rB; [PC]+2 →PC

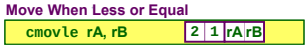
- As side effect, each of these instructions sets:
  - ZF flag to 1 if the result is zero, to 0 otherwise,
  - SF flag to the value of the most significant bit of the result,
  - OF flag to 1 if there is 2's complement overflow, 0 otherwise.

Y86-64 Reg-to-Reg Move Instructions

Move Unconditionally



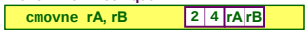
Move When Less or Equal



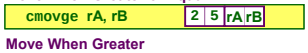
Move When Less

Move When Equal

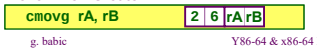
Move When Not Equal



Move When Greater or Equal



Move When Greater



- rrmovl instruction copies value from source register rA to destination register rB, i.e. [rA]→ 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 →PC

Y86-64 Immediate & Memory Move Instructions

Move Immediate to Register

V → rB

Move Register to Memory

[rA] → Memory[[rB]+D]

Move Memory to Register

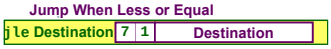
Memory[[rB]+D] → rA

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

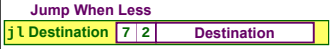
Y86-64 Jump Instructions

Jump Unconditionally

Jump When Less or Equal



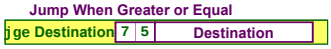
Jump When Less



Jump When Equal

Jump When Not Equal

Jump When Greater or Equal



Jump When Greater

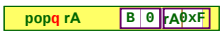
- jmp instruction copies value from its Destination field to PC, i.e. [Destiantion]→ PC
- Each other jump instruction copies a value from its Destination field to PC if jump condition is satisfied, otherwise PC value is incremented by 9 (the length of jump instruction)

Y86-64 Jump Instructions (cont.)

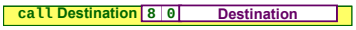
- jle instruction (jump when Less or Equal)
  - jump condition: (SF^OF) | ZF
- jl instruction (jump when Less)
  - jump condition: SF^OF
- je instruction (jump when Equal)
  - jump condition: ZF
- jne instruction (jump when Not Equal)
  - jump condition: ~ZF
- jge instruction (jump when Greater or Equal)
  - jump condition: ~(SF^OF)
- jg instruction (jump when Greater)
  - jump condition: ~(SF^OF) & ~ZF
- Move conditions are the same as these jump conditions.

Y86-64 Stack Effecting Instructions

- Decrement %rsp by 8
- Store word from rA to memory at (%rsp)
- [PC] + 2 → PC



- Read word from memory at (%rsp)
- Save in rA
- Increment %rsp by 8
- [PC] + 2 → PC



- pushes value PC+9 onto stack
- [Destination] → PC

- pops value from stack and loads it into PC

- [PC] +1 → PC

Useful x86-64 Instructions

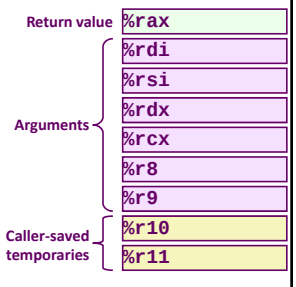
- cmpq instruction is used for setting of condition codes.
  - Format: cmpq src2, src1; like computing (a – b) without setting destination, where [src2] → b and [src1] → a
- testq instruction is also used for setting of condition codes.
  - Format: testq src2, src1; like computing a & b without setting destination, where [src2] → b and [src1] → 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 → %rk
  - c any integer, %ri, %rj, %rk any register, s=1,2,4 or 8; c1, %ri, %ri & s optional.

Examples of x86-64 Instructions

Format	*Computation*		
addq	*Src,Dest*	Dest = Dest + Src	
subq	*Src,Dest*	Dest = Dest – Src	
imulq	*Src,Dest*	Dest = Dest * Src	
salq	*Src,Dest*	Dest = Dest << Src	*Also called shlq*
sarq	*Src,Dest*	Dest = Dest >> Src	*Arithmetic*
shrq	*Src,Dest*	Dest = Dest >> Src	*Logical*
xorq	*Src,Dest*	Dest = Dest ^ Src	
andq	*Src,Dest*	Dest = Dest & Src	
orq	*Src,Dest*	Dest = Dest | Src	
incq	*Dest*	*Dest = Dest + 1*	
decq	*Dest*	*Dest = Dest – 1*	
negq	*Dest*	*Dest = – Dest*	
notq	*Dest*	*Dest = ~Dest*	

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 (1st, 2nd, 3rd,…, 6th) also caller-saved (if needed), since can be modified by called function
- %r10, %r11
  caller-saved (if neded), since can be modified by called function



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