Y86-64: Instruction Set Architecture

CMPU 224 – Computer Organization
Jason Waterman
Instruction Set Architecture

• Assembly Language View
  • Processor state
    • Registers, memory, …
  • Instructions
    • addq, pushq, ret, …
    • How instructions are encoded as bytes

• Layer of Abstraction
  • Above: how to program machine
    • Processor executes instructions in a sequence
  • Below: what needs to be built
    • Use a variety of tricks to make it run fast
    • E.g., execute multiple instructions simultaneously
Y86-64 Processor State

- **Program Registers**
  - 15 registers (omit %r15)
  - Each 64-bits long

- **Condition Codes**
  - Single-bit flags set by arithmetic and logical instructions
    - ZF: Zero
    - SF:Negative
    - OF: Overflow

- **Program Counter**
  - Indicates address of next instruction

- **Program Status**
  - Indicates either normal operation or some error condition

- **Memory**
  - Byte-addressable storage array
  - Words stored in little-endian byte order
Y86-64 Instructions

• Largely a subset of x86-64 instructions
• Only 8-byte integer operations
• Format
  • 1–10 bytes of information read from memory
  • Can determine instruction length from first byte
  • Not as many instruction types, and simpler encoding than with x86-64
Y86-64 Instruction Set

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>cmovXX rA, rB</td>
<td>2</td>
<td>fn</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irmovq V, rB</td>
<td>3</td>
<td>0</td>
<td>F</td>
<td>rB</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rmmovq rA, D(rB)</td>
<td>4</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mrmovq D(rB), rA</td>
<td>5</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPq rA, rB</td>
<td>6</td>
<td>fn</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jXX Dest</td>
<td>7</td>
<td>fn</td>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>call Dest</td>
<td>8</td>
<td>0</td>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushq rA</td>
<td>A</td>
<td>0</td>
<td>rA</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popq rA</td>
<td>B</td>
<td>0</td>
<td>rA</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Y86-64 Instruction Set

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovXX rA, rB</td>
<td>2</td>
<td>fn</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irmovq V, rB</td>
<td>3</td>
<td>0</td>
<td>F</td>
<td>rB</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rmmovq rA, D(rB)</td>
<td>4</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mrmmovq D(rB), rA</td>
<td>5</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPq rA, rB</td>
<td>6</td>
<td>fn</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jXX Dest</td>
<td>7</td>
<td>fn</td>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>call Dest</td>
<td>8</td>
<td>0</td>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushq rA</td>
<td>A</td>
<td>0</td>
<td>rA</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popq rA</td>
<td>B</td>
<td>0</td>
<td>rA</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `rrmovq` at byte 2, offset 0
- `cmovle` at byte 2, offset 1
- `cmovl` at byte 2, offset 2
- `cmove` at byte 2, offset 3
- `cmovne` at byte 2, offset 4
- `cmovge` at byte 2, offset 5
- `cmovg` at byte 2, offset 6

3/27/2023

CMPU 224 -- Computer Organization
### Y86-64 Instruction Set

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>0 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>1 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovXX rA, rB</td>
<td>2 fn rA rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irmovq V, rB</td>
<td>3 0 F rB V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rmmovq rA, D(rB)</td>
<td>4 0 rA rB D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mrmmovq D(rB), rA</td>
<td>5 0 rA rB D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPq rA, rB</td>
<td>6 fn rA rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jXX Dest</td>
<td>7 fn Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>call Dest</td>
<td>8 0 Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>9 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushq rA</td>
<td>A 0 rA F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popq rA</td>
<td>B 0 rA F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- `rrmovq` 2 0
- `cmovle` 2 1
- `cmovl` 2 2
- `cmovl` 2 3
- `cmovne` 2 4
- `cmovge` 2 5
- `cmovg` 2 6

- `addq` 6 0
- `subq` 6 1
- `andq` 6 2
- `xorq` 6 3
# Y86-64 Instruction Set

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmove</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovl</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovge</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovg</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovne</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovl</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovge</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovg</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovle</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmpu224</td>
<td>3/27/2023</td>
<td>Computer Organization</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Instructions
- **pushq rA**: A 0 rA F
- **popq rA**: B 0 rA F
- **addq**: 6 0
- **subq**: 6 1
- **andq**: 6 2
- **xorq**: 6 3
- **call Dest**: 8 0 Dest
- **ret**: 9 0
- **jmp**: 7 0
- **jle**: 7 1
- **jl**: 7 2
- **je**: 7 3
- **jne**: 7 4
- **jge**: 7 5
- **jg**: 7 6
- **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

3/27/2023
## Encoding Registers

- Each register has 4-bit ID
- Same encoding as in x86-64
- Register ID 15 (0xF) indicates “no register”
- Will use this in our hardware design in multiple places

<table>
<thead>
<tr>
<th>Register</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rax</td>
<td>0</td>
</tr>
<tr>
<td>%rcx</td>
<td>1</td>
</tr>
<tr>
<td>%rdx</td>
<td>2</td>
</tr>
<tr>
<td>%rbx</td>
<td>3</td>
</tr>
<tr>
<td>%rsp</td>
<td>4</td>
</tr>
<tr>
<td>%rbp</td>
<td>5</td>
</tr>
<tr>
<td>%rsi</td>
<td>6</td>
</tr>
<tr>
<td>%rdi</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Register</td>
<td>F</td>
</tr>
</tbody>
</table>

### Byte Instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td>0 0</td>
</tr>
<tr>
<td>nop</td>
<td>1 0</td>
</tr>
<tr>
<td>pushq rA</td>
<td>A 0 F</td>
</tr>
<tr>
<td>popq rA</td>
<td>B 0 F</td>
</tr>
<tr>
<td>cmovX X A, rB</td>
<td>2 fn rA rB</td>
</tr>
<tr>
<td>irmovq V, rB</td>
<td>3 0 F rB V</td>
</tr>
<tr>
<td>rmmovq rA, D(rB)</td>
<td>4 0 rA rB D</td>
</tr>
<tr>
<td>rmmovq D(rB), rA</td>
<td>5 0 rA rB D</td>
</tr>
<tr>
<td>OPq rA, rB</td>
<td>6 fn rA rB</td>
</tr>
<tr>
<td>jXX Dest</td>
<td>7 fn Dest</td>
</tr>
<tr>
<td>call Dest</td>
<td>8 0 Dest</td>
</tr>
<tr>
<td>ret</td>
<td>9 0</td>
</tr>
<tr>
<td>ret</td>
<td>9 0</td>
</tr>
</tbody>
</table>

3/27/2023

CMPU 224 -- Computer Organization
Instruction Example

• Addition Instruction
  • Add value in register rA to that in register rB
    • Store result in register rB
    • Note that Y86-64 only allows addition to be applied to register data
  • Set condition codes based on result
    • e.g., `addq %rax, %rsi` Encoding: 60 06
  • Two-byte encoding
    • First indicates instruction type
    • Second gives source and destination registers
Arithmetic and Logical Operations

- Refer to generically as “OPq”
- Encodings differ only by “function code”
  - Low-order 4 bits in first instruction word
- Set condition codes as side effect

**Instruction Code**

### Add

addq rA, rB

**Function Code**

6 0  rA  rB

### Subtract (rA from rB)

subq rA, rB

6 1  rA  rB

### And

andq rA, rB

6 2  rA  rB

### Exclusive-Or

xorq rA, rB

6 3  rA  rB
Move Operations

- Like the x86-64 `movq` instruction
- Simpler format for memory addresses
- Give different names to keep them distinct
## Move Instruction Example

### X86-64

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>movq $0xabcd, %rdx</code></td>
<td>30 F2 cd ab 00 00 00 00 00 00</td>
</tr>
<tr>
<td><code>movq %rsp, %rbx</code></td>
<td>20 43</td>
</tr>
<tr>
<td><code>movq -12(%rbp),%rcx</code></td>
<td>50 15 f4 ff ff ff ff ff ff ff ff</td>
</tr>
<tr>
<td><code>movq %rsi,0x41c(%rsp)</code></td>
<td>40 64 1c 04 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

### Y86-64

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>irmovq $0xabcd, %rdx</code></td>
<td>1F F2 cd ab 00 00 00 00 00 00</td>
</tr>
<tr>
<td><code>rrmovq %rsp, %rbx</code></td>
<td>20 43</td>
</tr>
<tr>
<td><code>rmrmovq -12(%rbp),%rcx</code></td>
<td>50 15 f4 ff ff ff ff ff ff ff ff ff</td>
</tr>
<tr>
<td><code>rmmovq %rsi,0x41c(%rsp)</code></td>
<td>40 64 1c 04 00 00 00 00 00 00</td>
</tr>
</tbody>
</table>

- **Little-endian**
- **Two’s complement**
## Conditional Move Instructions

### Move Unconditionally

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>rrmovq rA, rB</code></td>
<td>2 0 rA rB</td>
</tr>
</tbody>
</table>

### Move When Less or Equal

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmovle rA, rB</code></td>
<td>2 1 rA rB</td>
</tr>
</tbody>
</table>

### Move When Less

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmovl rA, rB</code></td>
<td>2 2 rA rB</td>
</tr>
</tbody>
</table>

### Move When Equal

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmove rA, rB</code></td>
<td>2 3 rA rB</td>
</tr>
</tbody>
</table>

### Move When Not Equal

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmovne rA, rB</code></td>
<td>2 4 rA rB</td>
</tr>
</tbody>
</table>

### Move When Greater or Equal

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmovge rA, rB</code></td>
<td>2 5 rA rB</td>
</tr>
</tbody>
</table>

### Move When Greater

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cmovg rA, rB</code></td>
<td>2 6 rA rB</td>
</tr>
</tbody>
</table>

- Refer to generically as “`cmovXX`”
- Encodings differ only by “function code”
- Based on values of condition codes
- Variants of `rrmovq` instruction
  - (Conditionally) copy value from source to destination register
Jump Instructions

• Refer to generically as “jXX”
• Encodings differ only by “function code” fn
• Based on values of condition codes
• Same as x86-64 counterparts
• Encode full destination address
  • Unlike PC-relative addressing seen in x86-64
Jump Instructions

Jump Unconditionally

jmp Dest 7 0 Dest

Jump When Less or Equal

jle Dest 7 1 Dest

Jump When Less

jl Dest 7 2 Dest

Jump When Equal

je Dest 7 3 Dest

Jump When Not Equal

jne Dest 7 4 Dest

Jump When Greater or Equal

jge Dest 7 5 Dest

Jump When Greater

jg Dest 7 6 Dest
Y86-64 Program Stack

- Region of memory holding program data
- Used in Y86-64 (and x86-64) for supporting procedure calls
- Stack top indicated by %rsp
  - Address of top stack element
- Stack grows toward lower addresses
  - Top element is at highest address in the stack
  - When pushing, must first decrement stack pointer
  - After popping, increment stack pointer
Stack Operations

- **pushq rA**
  - Decrement `%rsp` by 8
  - Store word from rA to memory at `%rsp`
  - Like x86-64

- **popq rA**
  - Read word from memory at `%rsp`
  - Save in rA
  - Increment `%rsp` by 8
  - Like x86-64
Subroutine Call and Return

- Push address of next instruction onto stack
- Start executing instructions at Dest
- Like x86-64

- Pop value from stack
- Use as address for next instruction
- Like x86-64
Miscellaneous Instructions

- **nop**
  - Don’t do anything

- **halt**
  - Stop executing instructions
  - x86-64 has comparable instruction, but it can’t be executed in user mode
  - We will use it to stop the simulator
  - Encoding ensures that program hitting memory initialized to zero will halt
Status Conditions

- Normal operation
- Halt instruction encountered
- Bad address (either instruction or data) encountered
- Invalid instruction encountered

Desired Behavior
- If AOK, keep going
- Otherwise, stop program execution

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOK</td>
<td>1</td>
</tr>
<tr>
<td>HLT</td>
<td>2</td>
</tr>
<tr>
<td>ADR</td>
<td>3</td>
</tr>
<tr>
<td>INS</td>
<td>4</td>
</tr>
</tbody>
</table>
Writing Y86-64 Code

• Can try to Use C Compiler
  • Write code in C
  • Compile for x86-64 with `gcc -Og -S`
  • Transliterate into Y86-64
  • *Modern compilers make this more difficult*

• Coding Example
  • Find the number of elements in null-terminated list
    ```
    int len1(int a[]);
    ```
    
    | a   | 5043 | 6125 | 7395 | 0 |
    |-----|------|------|------|---|
    |     |      |      |      | 3 |
    ```
Y86-64 Code Generation Example

• First Try
  • Write typical array code

```c
/* Find number of elements in null-terminated list */
long len(long a[])
{
    long len;
    for (len = 0; a[len]; len++);
    return len;
}
```

• Compile with `gcc -Og -S`

• Problem
  • Hard to do array indexing on Y86-64
    • Since don’t have scaled addressing modes

```asm
len:
    movl $0, %eax
.L3:
    cmpq $0, (%rdi,%rax,8)
    je .L2
    addq $1, %rax
    jmp .L3
.L2:
    ret
```

3/27/2023
CMPU 224 -- Computer Organization
• Second try
  • Write C code that mimics expected Y86-64 code

```c
long len(long a[]) {
    long val = *a;
    long len = 0;
    while (val) {
        a++;
        len++;
        val = *a;
    }
    return len;
}
```

```assembly
len:
    movq (%rdi), %rdx  # val = *a
    movl $0, %eax      # len = 0
.L3:
    testq %rdx, %rdx   # while(val)
    je .L2             # while(val)
    addq $8, %rdi      # a++
    addq $1, %rax      # len++
    movq (%rdi), %rdx  # val = *a
    jmp .L3            # jump to while test
.L2:
    ret                 # return len
```

**Register Use**
- `%rdi` - `a`
- `%rax` - `len`
- `%rdx` - `val`
Y86-64 Code Generation Example #3

```assembly
len:
    movq (%rdi), %rdx
    movl $0, %eax
.L3:
    testq %rdx, %rdx
    je .L2
    addq $8, %rdi
    addq $1, %rax
    movq (%rdi), %rdx
    jmp .L3
.L2:
    rep ret

len:
    irmovq $1, %r8          # Constant 1
    irmovq $8, %r9          # Constant 8
    mrmovq (%rdi), %rdx     # val = *a
    irmovq $0, %rax         # len = 0

    test:
    testq %rdx, %rdx         # Test val
    je .L2                   # If zero, goto Done
    addq $8, %rdi
    addq $1, %rax
    movq (%rdi), %rdx
    jmp .L3
.L2:
    rep ret

done:
    ret
```

<table>
<thead>
<tr>
<th>Register</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>a</td>
</tr>
<tr>
<td>%rax</td>
<td>len</td>
</tr>
<tr>
<td>%rdx</td>
<td>val</td>
</tr>
<tr>
<td>%r8</td>
<td>1</td>
</tr>
<tr>
<td>%r9</td>
<td>8</td>
</tr>
</tbody>
</table>
Y86-64 Sample Program Structure #1

- Program starts at address 0
- Must set up stack
  - Where located
  - Make sure don’t overwrite code!
- Must initialize data
  - See next slide

# Initialization

```
.pos 0     # Execution begins at address 0
irmovq stack, %rsp # Set up stack pointer
call main      # Execute main program
halt

.align 8      # Program data
array:
  . . .

main:        # Main function
  . . .
call len
  . . .

len:         # Length function
  . . .

.pos 0x200   # Placement of stack
stack:
```

3/27/2023
Y86-64 Program Structure #2

- Must initialize data
  - Can use symbolic names
- Set up call to len
  - Follow x86-64 procedure conventions
- Push array address as argument

```assembly
# Initialization
  .pos 0    # Execution begins at address 0
  irmovq stack, %rsp # Set up stack pointer

  call main    # Execute main program
  halt

# Array of 4 elements + terminating 0
  .align 8
array:
  .quad 0x00d00d00d00d00d
  .quad 0x00c00c00c00c0
  .quad 0xb00b00b00b00b
  .quad 0xa00a00a00a00
  .quad 0
main:
  irmovq array, %rdi
  call len
  ret
...
  .pos 0x200    # Placement of stack
stack:
```

3/27/2023
Assembling Y86-64 Programs (yas)

• Generates “object code” file len.yo
  • Looks like disassembler output

Linux> yas len.ys

0x054:
0x054: 30f801000000000000000000 | irmovq $1, %r8    # Constant 1
0x05e: 30f908000000000000000000 | irmovq $8, %r9    # Constant 8
0x068: 502700000000000000000000 | mrmovq (%rdi), %rdx # val = *a
0x072: 30f000000000000000000000 | irmovq $0, %rax    # len = 0
0x07c:  | test:
0x07c: 6222  |     andq %rdx, %rdx   # Test val
0x07e: 739e00000000000000000000 | je done            # If zero, goto Done
0x087: 6097  |     addq %r9, %rdi   # a++
0x089: 6080  |     addq %r8, %rax   # len++
0x08b: 502700000000000000000000 | mrmovq (%rdi), %rdx # val = *a
0x095: 707c00000000000000000000 |   jmp test         # Jump to test
0x09e: | done:
0x09e: 90    |     ret
Simulating Y86-64 Programs (yis)

- Instruction set simulator
  - Computes effect of each instruction on processor state
  - Prints changes in state from original

```
Linux> yis len.yo
```

Stopped in 37 steps at PC = 0x13. Status 'HLT', CC Z=1
S=0 O=0
Changes to registers:
%rax:  0x0000000000000000 0x0000000000000004
%rsp:  0x0000000000000000 0x0000000000000200
%rdi:  0x0000000000000000 0x0000000000000038
%r8:   0x0000000000000000 0x0000000000000001
%r9:   0x0000000000000000 0x0000000000000008

Changes to memory:
0x01f0: 0x0000000000000000 0x0000000000000053
0x01f8: 0x0000000000000000 0x0000000000000013
Y86-64 Instruction Set

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>halt</td>
<td></td>
<td></td>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nop</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovXX rA, rB</td>
<td>2</td>
<td>fn</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irmovq V, rB</td>
<td>3</td>
<td>0</td>
<td>F</td>
<td>rB</td>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rmmovq rA, D(rB)</td>
<td>4</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mrmmovq D(rB), rA</td>
<td>5</td>
<td>0</td>
<td>rA</td>
<td>rB</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OPq rA, rB</td>
<td>6</td>
<td>fn</td>
<td>rA</td>
<td>rB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jXX Dest</td>
<td>7</td>
<td>fn</td>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>call Dest</td>
<td>8</td>
<td>0</td>
<td>Dest</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td>9</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pushq rA</td>
<td>A</td>
<td>0</td>
<td>rA</td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>popq rA</td>
<td>B</td>
<td>0</td>
<td>rA</td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rrmovq</td>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovle</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovl</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmove</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovne</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovge</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmovg</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

%rax | 0
%rcx | 1
%rdx | 2
%rbx | 3
%rsp | 4
%rbp | 5
%rsi | 6
%rdi | 7
%r8  | 8
%r9  | 9
%r10 | A
%r11 | B
%r12 | C
%r13 | D
%r14 | E
No Register | F