# Basic Assembly

## The REP Prefixes

Assembly language programming
By xorpd

# Objectives

- We will study the instructions:
  - SCAS
  - COMPS
- We will learn about the prefixes:
  - REP
  - REPZ, REPNZ

# Example

- Challenge: “clearing” an array of bytes.
- Example:

```
my_arr    db    ARR_LEN dup (?)
```

```
  mov    edi,my_arr
  mov    ecx,ARR_LEN
  xor    al,al

next_element:
  stosb
  loop   next_element
```

- Doesn’t work well if ARR_LEN = 0 :(
  - We can fix it!

# Example

- Challenge: “clearing” an array of bytes.

- Example:

```
my_arr    db    ARR_LEN dup (?)
```

```
  mov    edi,my_arr
  mov    ecx,ARR_LEN
  xor    al,al

  jecxz  finish
next_element:
  stosb
  loop   next_element
finish:
```

# Example (Cont.)

- Instead, we can use the REP prefix:

```
my_arr    db    ARR_LEN dup (?)
```

```
  mov    edi,my_arr
  mov    ecx,ARR_LEN
  xor    al,al

  rep stosb
```

- This code will behave exactly like the previous one.

# The REP Prefix

- **REP** <string instruction>
  - Works with a few predefined instructions (stos,lods,movs,…)

- Repeats the string instruction **ecx** times.
- ecx will become zero in the end of the compound instruction.

```
  jecxz  finish
one_iter:
  <string instruction>
  loop   one_iter
finish:
```

- Examples:
  - rep stosb ; Initialize an array.
  - rep movsd ; Copy data.

# REP MOVSD Example

- Copying an array:

```
src_arr    db    ARR_LEN dup (?)
dest_arr   db    ARR_LEN dup (?)
```

```
  mov    esi,src_arr
  mov    edi,dest_arr
  mov    ecx,ARR_LEN

  rep movsb
```

# REP MOVSD Example

- Copying an array:

```
src_arr     db     ARR_LEN dup (?)
dest_arr    db     ARR_LEN dup (?)
```

```
  mov     esi,src_arr
  mov     edi,dest_arr
  mov     ecx,ARR_LEN

  rep movsb
```

Do ecx times:

- $[edi] \leftarrow [esi]$
- $esi \leftarrow esi + 1$
- $edi \leftarrow edi + 1$

# SCAS

- SCAS – Scan String
- Some forms:
  - SCASB (Byte)
    - Compare $al$ with $[edi]$. Set flags accordingly.
    - Advance $edi$ 1 byte (According to DF).
  - SCASW (Word)
    - Compare $ax$ with $[edi]$. Set flags accordingly.
    - Advance $edi$ 2 bytes (According to DF).
  - SCASD (Dword)
    - Compare $eax$ with $[edi]$. Set flags accordingly.
    - Advance $edi$ 4 bytes (According to DF).

# SCAS Example

- Finding the Null terminator of a string.

```
a_str    db    'I am a string',0
```

```
  mov    edi,a_str
  xor    al,al ; Null.

next_char:
  scasb
  jnz    next_char
```

# SCAS Example

- Finding the Null terminator of a string.

```
a_str    db    'I am a string',0
```

```
  mov    edi,a_str
  xor    al,al ; Null.

next_char:
  scasb
  jnz    next_char
```

```
  mov    edi,a_str
  xor    al,al ; Null.

next_char:
  cmp    al,byte [edi]
  lea    edi,[edi+1]
  jnz    next_char
```

# SCAS Example

- Finding the Null terminator of a string.

```
a_str    db    'I am a string',0
```

```
  mov    edi,a_str
  xor    al,al ; Null.

next_char:
  scasb
  jnz    next_char
```

```
  mov    edi,a_str
  xor    al,al ; Null.

next_char:
  cmp    al,byte [edi]
  lea    edi,[edi+1]
  jnz    next_char
```

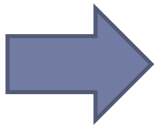
Lea doesn't change the flags!

# SCAS Example (Cont.)

- Finding the Null terminator of a string.
- What if we never find the Null terminator?
  - We want a failsafe mechanism.

```
  mov     edi,a_str
  xor     al,al ; Null.

next_char:
  scasb
  jnz     next_char
```



```
  mov     ecx,MAX_STR
  mov     edi,a_str
  xor     al,al ; Null.

next_char:
  jecxz   exit_loop
  dec     ecx
  scasb
  jnz     next_char
exit_loop:
```

- Two termination conditions:
  - $ZF == 1$ ; Finding the character.
  - $ecx == 0$ ; Failsafe mechanism.

# SCAS Example (Cont.)

- Finding the Null terminator of a string.
- Instead, we can use the **REPNZ** prefix!

```
  mov     ecx,MAX_STR
  mov     edi,a_str
  xor     al,al ; Null.

next_char:
  jecxz   exit_loop
  dec     ecx
  scasb
  jnz     next_char
exit_loop:
```

```
mov     ecx,MAX_STR
mov     edi,a_str
xor     al,al ; Null.

repnz scasb
```

# SCAS Example (Cont.)

- Finding the Null terminator of a string.
- Instead, we can use the **REPNZ** prefix!

```
  mov     ecx,MAX_STR
  mov     edi,a_str
  xor     al,al ; Null.

next_char:
  jecxz   exit_loop
  dec     ecx
  scasb
  jnz     next_char
exit_loop:
```

```
  mov     ecx,MAX_STR
  mov     edi,a_str
  xor     al,al ; Null.

  repnz scasb
```

# REPZ, REPNZ

- REPNZ <string instruction>
  - "Repeat as long as not zero". (As long as ZF=0)

- Repeats the string instruction as long as:
  - The zero flag is cleared.
  - ecx is not zero.

- In every iteration:
  - If $ecx == 0$:
    - Terminate
  - Decrease $ecx$ by $1$.
  - Execute <string instruction>
  - If ZF = 1:
    - Terminate

```
next_iter:
  jecxz  exit_loop
  dec    ecx
  <string instruction>
  jnz    next_iter
exit_loop:
```

# REPZ, REPNZ

- REPZ <string instruction>
  - "Repeat as long as zero". (As long as ZF=1)

- Repeats the string instruction as long as:
  - The zero flag is set.
  - ecx is not zero.

- In every iteration:
  - If $ecx == 0$:
    - Terminate
  - Decrease $ecx$ by $1$.
  - Execute <string instruction>
  - If ZF = 0:
    - Terminate

```
next_iter:
  jecxz  exit_loop
  dec    ecx
  <string instruction>
  jz     next_iter
exit_loop:
```

# REPZ, REPNZ

- Summary of termination conditions:

| Prefix | Flags termination condition | ecx termination condition |
|---|---|---|
| REP | None | ecx = 0 |
| REPZ / REPE | ZF = 0 | ecx = 0 |
| REPNZ / REPNE | ZF = 1 | ecx = 0 |

- REPZ, REPNZ might stop as a result of one of two termination conditions.
  - We might have to further investigate after the repetition ends.

# Checking Termination Cause

- REPNZ - How can we know what caused the repetitions to stop?
  - ecx = 0  We have reached MAX_STR
  - ZF = 1  A null terminator character was found.
- We check!

```
  mov     ecx,MAX_STR
  mov     edi,a_str
  xor     al,al ; Null.

  repnz scasb
```

```
  jnz     null_not_found
  ; Null found.
  jmp     end_null_check
null_not_found:
  ; Null not found.
end_null_check:
  ...
```

# CMPS

- CMPS – Compare String Operands
- Some forms:
  - CMPSB (Byte)
    - Compare $[esi]$ with $[edi]$. Set flags accordingly.
    - Advance $esi$ and $edi$ 1 byte. (According to DF)
  - CMPSW (Word)
    - Compare $[esi]$ with $[edi]$. Set flags accordingly.
    - Advance $esi$ and $edi$ 2 bytes. (According to DF)
  - CMPSD (Dword)
    - Compare $[esi]$ with $[edi]$. Set flags accordingly.
    - Advance $esi$ and $edi$ 4 bytes. (According to DF)

# CMPS Example

- Checking equality of two buffers of the same known size:

```
buff1    dd    NUM_DW dup (?)
buff2    dd    NUM_DW dup (?)
```

```
    mov    esi,buff1
    mov    edi,buff2
    mov    ecx,NUM_DW

    repz cmpsd
```

```
    jz     buffers_equal
    ; Buffers differ:
    sub    edi,4
    sub    esi,4

    jmp    end_if
buffers_equal:
    ; Buffers are equal.
end_if:
```

# Summary

- Instructions:
  - SCAS – Scan String.
  - CMPS – Compare String operands.
- Prefixes:
  - REP
  - REPZ (REPE) , REPNZ (REPNE).
- Possible combinations:
  - `REP, REPZ, REPNZ`
  - `LODS, STOS, MOVS, SCAS, CMPS`
- Meaningful idioms to remember:
  - rep stos — Initializing an array.
  - rep movs — Copying data.
  - repnz scas — Searching a character.
  - repz cmps — Comparing buffers.