IA32 Supplement

CS211 Spring 2020

Segment Registers

- All indirect references can be specified relative to a *segment register*
    - These allowed older processors to use 16-bit pointers to have access to a 24-bit address space
    - The segment registers are `%cs`, `%ds`, `%ss`, `%es`, `%fs`, `%gs`
- Syntax: `%ds:(%esi)` — effective address given by `%esi` relative to `%ds`
    - These are normally implicit, based on the type of operation
    - Most modern OS set all the segment registers to 0
    - In particular, you can ignore any segment offsets in the bomb lab

String Instructions

- These are left over from the days when people would write assembly programs directly
- Designed for working with strings and string-like arrays
- Instead of having operands, they work with specific registers
- They are fairly complex; allowing a programmer or compiler to write a loop in a single instruction
    - This sort of complexity is a key difference between the CISC and RISC design philosophies
- We normally don't discuss these, but they started showing up in PA3 this semester

`movs` — Copy String

- `movsb` — copy a byte from the location indicated by `%esi` (the string source pointer) to the location indicated by `%edi` (the string destination register)
  - `movsw` and `movsl` move 2- and 4-byte data
  - objdump and GDB explicitly write `%esi` and `%edi` as arguments, but this is redundant; different operands cannot be specified
- `rep movsb` — copy a number of bytes given in `%ecx` (the counter register) from the address in `%esi` to the address in `%edi`
  - This actually performs `movsb` multiple times
  - After each move, it increments `%esi` and `%edi` and decrements `%ecx`
  - The loop halts when `%ecx` is 0

`stos` — Store String

- `stosb` — Copy `%al` to the memory location given by `%edi`
  - The same as `movb %al, (%edi)`
  - `stosw` — Copy `%ax` to `(%edi)`
  - `stosl` —Copy `%eax` to `(%edi)`
- `rep stos` — Copy `%al` to `%ecx` bytes starting at `(%edi)`
  - Useful to zero out a region of memory in one instruction

`scas` — Search String

- `scasb` — Compare the byte pointed to by `%esi` to `%al` and set condition codes
    - Equivalent to `cmpb (%esi), %al`
    - `scasw` compares a word with `%ax`
    - `scasl` compares a double word with `%eax`
- `repz scasb` — Compare up to `%ecx` bytes starting at `%esi` to `%al`, halting if the byte is not equal to `%al`
- `repnz scasb` — Similar, but halting when the byte is equal to `%al`
- `repe` and `repne` are synonyms for `repz` and `repnz`

`rep`/`repz`/`repnz`

- `rep`, `repz`, and `repnz` are technically separate instructions that affect the meaning of the next instruction
    - They are only meaningful when the next instruction is one of the string instructions
        - Some string instructions work with `rep`, others `repz` and `repnz`
        - In fact, `rep` and `repz` have the same binary representation
    - objdump and GDB usually present this and the subsequent instruction as single unit
- You may occasionally see `repz ret` in code — this is the same as `ret`
    - GCC uses this when it wants to insert an extra byte into the binary (for alignment reasons)

References

- Information about rep/repz/repnz
  - https://www.felixcloutier.com/x86/rep:repe:repz:repne:repnz
- General string instruction reference
  - https://docs.oracle.com/cd/E19455-01/806-3773/6jct9o0aq/index.html
- The deal with repz ret
  - https://repzret.org/p/repzret/