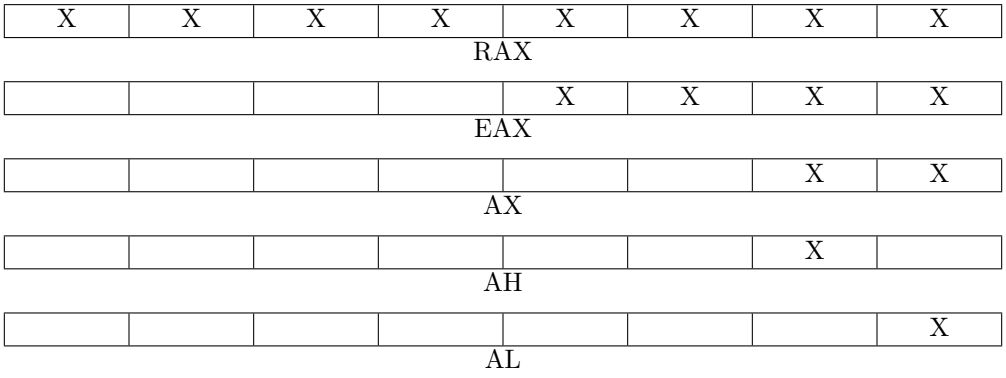
1 Rejestr RAX



2 Pozostałe rejestry

- RBX, EBX, BX, BH, BL
- RCX, ECX, CX, CH, CL
- RDX, EDX, DX, DH, DL
- RBP, EBP, BP, —, BPL
- RSP, ESP, SP, —, SPL *(RSP wskazuje na szczytowy element stosu)*
- RSI, ESI, SI, —, SIL
- RDI, EDI, DI, —, DIL
- R8, R8D, R8W, —, R8B
  ...
- R15, R15D, R15W, —, R15B

3 Wymagania ABI

Rejestry zachowywane przez funkcje: RBX, RBP, RSP, R12, R13, R14, R15

Kolejność parametrów: RDI, RSI, RDX, RCX, R8, R9

Wynik: RAX *(gdy wynik mieści się w 64 bitach)* lub RDX:RAX

Wyrównanie stosu: Na początku funkcji: RSP $\equiv_{16} 8$, czyli przed wywołaniem call: RSP $\equiv_{16} 0$

4 Wywołania systemowe w Linuxie (`syscall`)

Numer wywołania systemowego: RAX

Kolejność parametrów (inna!): RDI, RSI, RDX, R10, R8, R9

Wynik: RAX (jeśli $-4095 \leq \text{RAX} \leq -1$, to błąd i `errno` $= -\text{RAX}$)

Nadpisywane rejestry: RCX, R11 (oraz RAX na wynik)

Numery wywołań systemowych w Linuxie (64 bity): `http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/`

Specyfikacja wywołania systemowego (np. `sys_write`): w terminalu `man 2 write`

5 Na początku programu (bez biblioteki standardowej C)

- Wykonanie zaczyna się pod publiczną etykietą `_start`
- Na początku stos jest wyrównany $\equiv_{16} 0$ (inaczej niż w funkcjach!)
- 8-bajtowa zmienna `argc` pod adresem RSP (na szczycie stosu); wskaźniki na null-terminated teksty `argv[0]`, `argv[1]`, ... kolejno pod adresami RSP+8, RSP+16, ...
- Należy zakończyć wykonanie programu wywołaniem systemowym `sys_exit`

6 Przykład programu

```
global _start   ; etykieta _start (entry point programu) jest globalna

; stałe używane wewnątrz kodu programu
SYS_WRITE       equ 1           ; numer wywołania systemowego sys_write
SYS_EXIT        equ 60          ; numer wywołania systemowego sys_exit
STDOUT_FILENO   equ 1           ; numer deskryptora standardowego wyjścia

; dane tylko do odczytu
section .rodata

text_hi:        db "hi", `\n`   ; ciąg trzech bajtów 'h', 'i', `\n` (\n otoczone backtickami)

; kod wykonywalny
section .text

_start:
        ; Wypisujemy na standardowe wyjście (deskryptor 1) tekst 'hi'.
        ; Uwaga: operacje na 32-bitowych młodszych połówkach rejestrów zerują
        ; 32-bitowe starsze połówki rejestrów.
        mov     eax, SYS_WRITE
        mov     edi, STDOUT_FILENO
        mov     rsi, text_hi
        mov     edx, 3
        syscall

        ; Kończymy program z kodem błędu 0 (sukces)
        mov     eax, SYS_EXIT
        xor     edi, edi
        syscall
```