Under the Hood

Assembly, System Calls, and Hardware

David Sankel
Under the Hood
Assembly, System Calls, and Hardware

David Sankel | Principal Scientist
C++Now 2023
int chain(int a, int b, int c, int d) {
    int r = a + b + c + d;
    return r;
}

int unchained(int a, int b, int c, int d) {
    int r = (a + b) + (c + d);
    return r;
}
```c
int chain(int a, int b, int c, int d)
{
    int r = a + b + c + d;
    return r;
}

int unchained(int a, int b, int c, int d)
{
    int r = (a + b) + (c + d);
    return r;
}
```
```assembly
chain(int, int, int, int):
    add    esi, edi
    lea    eax, [rdx + rcx]
    add    eax, esi
    ret

unchained(int, int, int, int):
    add    esi, edi
    lea    eax, [rdx + rcx]
    add    eax, esi
    ret
```
```cpp
int chain(int a, int b, int c, int d)
{
    int r = a + b + c + d;
    return r;
}

int unchained(int a, int b, int c, int d)
{
    int r = (a + b) + (c + d);
    return r;
}
```

```assembly
chain(int, int, int, int):
    add   esi, edi
    lea   eax, [rdx + rcx]
    add   eax, esi
    ret

unchained(int, int, int, int):
    add   esi, edi
    lea   eax, [rdx + rcx]
    add   eax, esi
    ret
```
chain(int, int, int, int):
  add   esi, edi
  lea   eax, [rdx + rcx]
  add   eax, esi
  ret

unchained(int, int, int, int):
  add   esi, edi
  lea   eax, [rdx + rcx]
  add   eax, esi
  ret
chain(int, int, int, int, int):
  add    esi, edi
  lea    eax, [rdx + rcx]
  add    eax, esi
  ret

unchained(int, int, int, int, int):
  add    esi, edi
  lea    eax, [rdx + rcx]
  add    eax, esi
  ret
chain(int, int, int, int, int):
   add    esi, edi
   lea    eax, [rdx + rcx]
   add    eax, esi
   ret

unchained(int, int, int, int, int):
   add    esi, edi
   lea    eax, [rdx + rcx]
   add    eax, esi
   ret
chain(int, int, int, int, int):
  add   esi, edi
  lea   eax, [rdx + rcx]
  add   eax, esi
  ret

unchained(int, int, int, int, int):
  add   esi, edi
  lea   eax, [rdx + rcx]
  add   eax, esi
  ret
```cpp
int chain(int a, int b, int c, int d)
{
    int r = a + b + c + d;
    return r;
}

int unchained(int a, int b, int c, int d)
{
    int r = (a + b) + (c + d);
    return r;
}
```

```assembly
chain(int, int, int, int):
    add    esi, edi
    lea    eax, [rdx + rcx]
    add    eax, esi
    ret

unchained(int, int, int, int):
    add    esi, edi
    lea    eax, [rdx + rcx]
    add    eax, esi
    ret
```
```cpp
int chain(int a, int b, int c, int d) {
    int r = a + b + c + d;
    return r;
}

int unchained(int a, int b, int c, int d) {
    int r = (a + b) + (c + d);
    return r;
}
```
Don’t make performance claims without benchmarks.

(... unless you can show identical generated assembly. In that case, go right ahead)
Don’t make performance claims without benchmarks.

(... unless you can show identical generated assembly. In that case, go right ahead)
What is manual memory management?
What is manual memory management?

In computer science, **manual memory management** refers to the usage of manual instructions by the programmer to identify and deallocate unused objects, or garbage.

—Wikipedia

In assembly language, instructions are required to move information between the RAM and CPU.
Data transfer
Registers

CPU

RAM

2
6

2
4
Registers

8 bits

Intel 8008

A, B, C, D, E, H, L, PC
Registers

Intel 8008

- Accumulator
  - A
  - B
  - C
  - D
  - PC
- High-order byte
  - E
  - H
- Low-order byte
  - L
- Program Counter
Registers

Intel 8008

- A
- B
- C
- D
- E
- H
- L
- PC
Registers

Intel 8008

Intel 8086

AH AL
AX
BH BL
BX
CH CL
CX
DH DL
DX
SP
BP
SI
DI
IP

CPU
Registers

Intel 8008

Intel 806

AH AL
AX
BH BL
BX
CH CL
CX
DH DL
DX
SP
BP
SI
DI
IP

D High and D Low 8bit registers
Registers

Intel 8008

Intel 8086

DX 16 bit register. X is a placeholder as in version 3.x or x86.
Registers

Intel 8008

- A
- B
- C
- D
- E
- H
- L
- PC

Intel 8086

- AH AL
- AX AL
- BH BL
- BX BL
- CH CL
- CX CL
- DH DL
- DX DL
- SP
- BP
- SI
- DI
- IP

- Accumulator
- Base
- Count
- Data
- Stack Pointer
- Base Pointer
- Source Index
- Destination Index
- Instruction Pointer
Registers

Intel 8086

- A
- B
- C
- D
- PC

Intel 8086

- AH AL
- AX
- BH BL
- BX
- CX
- CH CL
- DX
- DH DL
- DX
- IP

Intel 80386

- AH AL
- AX
- BH BL
- BX
- CX
- CH CL
- DX
- DH DL
- EDX
- SP
- ESP
- BP
- EBP
- SI
- ESI
- DI
- EDI
- IP
- EIP

Extended Accumulator X

CPU
# Registers

## Intel 8008

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>PC</th>
</tr>
</thead>
</table>

## AMD x86-64

<table>
<thead>
<tr>
<th>AH</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP</td>
<td>SP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BH</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBP</td>
<td>BP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CH</th>
<th>DL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESI</td>
<td>SI</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EAX</th>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAX</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EBX</th>
<th>BX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBX</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ECX</th>
<th>CX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCX</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EDX</th>
<th>DX</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDX</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EDI</th>
<th>DI</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDI</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EIP</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIP</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R8D</th>
<th>R8B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R9D</th>
<th>R9B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R10D</th>
<th>R10B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R10</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R11D</th>
<th>R11B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R11</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R12D</th>
<th>R12B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R12</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R13D</th>
<th>R13B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R13</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R14D</th>
<th>R14B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R14</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R15D</th>
<th>R15B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R15</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R15W</th>
<th>R15B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R15D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R14W</th>
<th>R14B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R14D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R13W</th>
<th>R13B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R13D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R12W</th>
<th>R12B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R12D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R11W</th>
<th>R11B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R11D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R10W</th>
<th>R10B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R10D</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R9W</th>
<th>R9B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R8W</th>
<th>R8B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R8</td>
<td></td>
</tr>
</tbody>
</table>
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
<table>
<thead>
<tr>
<th>RAX</th>
<th>EAX</th>
<th>AX</th>
<th>RSP</th>
<th>SPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>RBX</td>
<td>EBX</td>
<td>BX</td>
<td>RBP</td>
<td>BPL</td>
</tr>
<tr>
<td>RCX</td>
<td>ECX</td>
<td>CX</td>
<td>RSI</td>
<td>SIL</td>
</tr>
<tr>
<td>RDX</td>
<td>EDX</td>
<td>DX</td>
<td>RDI</td>
<td>DIL</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>R8</th>
<th>R8B</th>
<th>R8D</th>
<th>R8W</th>
<th>R12</th>
<th>R12B</th>
</tr>
</thead>
<tbody>
<tr>
<td>R9</td>
<td>R9B</td>
<td>R9D</td>
<td>R9W</td>
<td>R13</td>
<td>R13B</td>
</tr>
<tr>
<td>R10</td>
<td>R10B</td>
<td>R10D</td>
<td>R10W</td>
<td>R14</td>
<td>R14B</td>
</tr>
<tr>
<td>R11</td>
<td>R11B</td>
<td>R11D</td>
<td>R11W</td>
<td>R15</td>
<td>R15B</td>
</tr>
</tbody>
</table>

add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
### AMD x86-64

<table>
<thead>
<tr>
<th>AL</th>
<th>AH</th>
<th>EAX</th>
<th>AX</th>
<th>ESP</th>
<th>SP</th>
<th>RSP</th>
<th>BPL</th>
<th>RBP</th>
<th>RAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>BL</td>
<td>BH</td>
<td>EBX</td>
<td>BX</td>
<td>EBP</td>
<td>BP</td>
<td>RBP</td>
<td>RSP</td>
<td>EAX</td>
<td>AL</td>
</tr>
<tr>
<td>DL</td>
<td>CH</td>
<td>ECX</td>
<td>CX</td>
<td>ESI</td>
<td>SI</td>
<td>RSI</td>
<td>RBP</td>
<td>EBX</td>
<td>BL</td>
</tr>
<tr>
<td>DL</td>
<td>CH</td>
<td>EDX</td>
<td>DX</td>
<td>EDI</td>
<td>DI</td>
<td>RDI</td>
<td>RSP</td>
<td>ECX</td>
<td>CH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>R8</td>
<td>R8W</td>
<td>R12D</td>
<td>R12W</td>
<td>R12</td>
<td>RBP</td>
<td>R8B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R9</td>
<td>R9W</td>
<td>R13D</td>
<td>R13W</td>
<td>R13</td>
<td>RSP</td>
<td>R9B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R10</td>
<td>R10W</td>
<td>R14D</td>
<td>R14W</td>
<td>R14</td>
<td>EAX</td>
<td>R10B</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>R11</td>
<td>R11W</td>
<td>R15D</td>
<td>R15W</td>
<td>R15</td>
<td>EAX</td>
<td>R11B</td>
<td></td>
</tr>
</tbody>
</table>

```
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
```
<table>
<thead>
<tr>
<th>AH</th>
<th>AL</th>
<th>AH</th>
<th>AL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>AX</td>
<td>AX</td>
<td>AX</td>
</tr>
<tr>
<td>ESP</td>
<td>SP</td>
<td>ESP</td>
<td>SP</td>
</tr>
</tbody>
</table>

```
add esi, edi
lea eax [rdx + rcx]
add eax, esi
ret
```
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
Intel 64 and IA-32 Architectures Software Developer’s Manual

- 5062 pages (C++ standard is 2120)
- Excellent reference for Intel assembly
- There's a PDF freely available
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
**LEA—Load Effective Address**

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Compat/Leg Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8D /r</td>
<td>LEA r16,m</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Store effective address for m in register r16.</td>
</tr>
<tr>
<td>8D /r</td>
<td>LEA r32,m</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Store effective address for m in register r32.</td>
</tr>
<tr>
<td>REX.W + 8D /r</td>
<td>LEA r64,m</td>
<td>RM</td>
<td>Valid</td>
<td>N.E.</td>
<td>Store effective address for m in register r64.</td>
</tr>
</tbody>
</table>

**Instruction Operand Encoding**

<table>
<thead>
<tr>
<th>Op/En</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Operand 3</th>
<th>Operand 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RM</td>
<td>ModRM:reg (w)</td>
<td>ModRM:r/m (r)</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

**Description**

Computes the effective address of the second operand (the source operand) and stores it in the first operand (destination operand). The source operand is a memory address (offset part) specified with one of the processors addressing modes; the destination operand is a general-purpose register. The address-size and operand-size attributes affect the action performed by this instruction, as shown in the following table. The operand-size attribute of the instruction is determined by the chosen register; the address-size attribute is determined by the attribute of the code segment.
Compiler, you are acting absurd

- There are add instructions y’know
- You’re using 64 bit registers instead of 32 bit
- This is just weird

```
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
```
That's the only way you can do that with a 3-byte instruction
Optimized compiler generated assembly

- Fast
- Not always pretty
- Works

```
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
```
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
How do functions work?

add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret

Call a function
- Set memory/registers to argument values according to a calling convention
- Make backups of registers you don't want clobbered
- call (put RIP on the stack and set RIP to the function address)
## System V AMD64 ABI (Linux/macOS)

### Argument 1
- `R8B`
- `R9B`
- `R10B`
- `R11B`
- `R12B`
- `R13B`
- `R14B`
- `R15B`

### Argument 2
- `R12`
- `R13`
- `R14`
- `R15`

### Argument 3 / Return 2
- `R8W`
- `R9W`
- `R10W`
- `R11W`

### Argument 4
- `EAX`
- `EBX`
- `ECX`
- `EDX`

### Argument 5
- `AX`
- `BX`
- `CX`
- `DX`

### Return 1
- `ESP`
- `SP`
- `RSP`
- `ESP`
- `SP`
- `RSP`

### Return 2
- `SI`
- `DI`
- `RDI`
- `SI`
- `DI`
- `RDI`

### RIP
- `EIP`
- `IP`
- `RIP`
- `EIP`
- `IP`
- `RIP`
System V AMD64 ABI (Linux/macOS)

```
add esi, edi
lea eax, [rdx + rcx]
add eax, esi
ret
```
Microsoft x64 calling convention
System V AMD64 ABI (Linux/macOS)

ARGUMENTS:
- Argument 1
- Argument 2
- Argument 3 / Return 2
- Argument 4
- Argument 5
- Argument 6

CALLER STORED:
- RAX
- RBX
- RCX
- RDX
- R8
- R9
- R10
- R11

CALLEE STORED:
- RSP
- RBP
- RSI
- RDI
- R12
- R13
- R14
- R15

RETURNS:
- Return 1
- Return 2
MOV

mov r10, 255  ; set register 10 to the number 255
mov r10, [255]  ; set register 10 to the number at address 255 in RAM
mov [255], r10  ; set RAM at address 255 to the number in register 10
mov r11, r10   ; set register 11's number to register 10's number
MOV

`mov r10, 255 ; set register 10 to the number 255`
`mov r10, [255] ; set register 10 to the number at address 255 in RAM`
`mov [255], r10 ; set RAM at address 255 to the number in register 10`
`mov r11, r10 ; set register 11's number to register 10's number`
MOV

- `mov r10, 255` ; set register 10 to the number 255
- `mov r10, [255]` ; set register 10 to the number at address 255 in RAM
- `mov [255], r10` ; set RAM at address 255 to the number in register 10
- `mov r11, r10` ; set register 11's number to register 10's number
**MOV**

- `mov r10, 255` ; set register 10 to the number 255
- `mov r10, [255]` ; set register 10 to the number at address 255 in RAM
- `mov [255], r10` ; set RAM at address 255 to the number in register 10
- `mov r11, r10` ; set register 11's number to register 10's number
MOV

mov r10, 255 ; set register 10 to the number 255
mov r10, [255] ; set register 10 to the number at address 255 in RAM
mov [255], r10 ; set RAM at address 255 to the number in register 10
mov r11, r10 ; set register 11's number to register 10's number
mov r10, 255
mov r10, [255]
mov [255], r10
mov r11, r10
mov r10, 255
mov r10, [255]
mov [255], r10
mov r11, r10

Write,

int main() { return 3; }

, in assembly.
Write, in assembly:

```c
int main() { return 3; }
```

```
mov r10, 255
mov r10, [255]
mov [255], r10
mov r11, r10
mov eax, 3
ret
```
What does this output?

```cpp
#include <iostream>

int main( int argc, char ** argv ) {
    std::cout << (void*)argv[argc] << std::endl;
}
```
What does this output?

#include <iostream>

int main( int argc, char ** argv ) {
    std::cout << (void*)argv[argc] << std::endl;
}

0
What does this do?

```cpp
#include <iostream>

int main( int argc, char ** argv ) {
    std::cout << argv[argc+1] << std::endl;
}
```
What does this do?

```c++
#include <iostream>

int main( int argc, char ** argv ) {
    std::cout << argv[argc+1] << std::endl;
}
```

SSH_TTY=/dev/pts/2
What does this do in Compiler Explorer?

```cpp
#include <iostream>

int main( int argc, char ** argv ) {
    std::cout << argv[argc+1] << std::endl;
}
```

#include <iostream>

int main( int argc, char ** argv ) {
    std::cout << argv[argc+1] << std::endl;
    std::cout << argv[argc+2] << std::endl;
    std::cout << argv[argc+3] << std::endl;
}

SSH_TTY=/dev/pts/2
PWD=/home/david/Documents/C++Now/2023/examples
MAIL=/var/spool/mail/david
#include <iostream>

int main( int argc, char ** argv ) {
    for(int i = argc+1; argv[i]; ++i)
        std::cout << argv[i] << std::endl;
}

SSH_TTY=/dev/pts/2
PWD=/home/david/Documents/C++Now/2023/examples
MAIL=/var/spool/mail/david
SSH_TTY=/dev/pts/2
PWD=/home/david/Documents/C++Now/2023/examples
MAIL=/var/spool/mail/david
XDG_DATA_DIRS=/home/david/.local/share/flatpak/exports/share:/home/david/Documents/C++Now/2023:
OLDPWD=/home/david/Documents/C++Now/2023
LANG=en_US.UTF-8
MAKEFLAGS=
MFLAGS=
XDG_SESSION_TYPE=tty
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SHLVL=1
MAKELEVEL=1
PATH=/home/david/Programs/ginger/bin:/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/lib/jvm/default/bin:/usr/bin/site_perl:/usr/bin/vendor_perl:/usr/bin/core_perl
MOTD_SHOWN=pam
SSH_CLIENT=fe80::f894:8d01:d0a4:16b%enp1s0 64412 22
LOGNAME=david
USER=david
DEBUGINFOD_URLS=https://debuginfod.archlinux.org
_==/usr/bin/make
MAKE_TERMOUT=/dev/pts/2
XDG_RUNTIME_DIR=/run/user/1000
VISUAL=vim
MAKE_TERMERR=/dev/pts/2
XDG_SESSION_CLASS=user
HOME=/home/david
HG=/usr/bin/hg
TERM=xterm
XDG_SESSION_ID=5
SSH_CONNECTION=fe80::f894:8d01:d0a4:16b%enp1s0 64412 fe80::2e27:d7ff:fe3b:680c%enp1s0 22
SHELL=/bin/bash
**ELF Loading**

<table>
<thead>
<tr>
<th>position</th>
<th>content</th>
<th>size (bytes) + comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack pointer</td>
<td>argc = number of args</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[0] (pointer)</td>
<td>4 (program name)</td>
</tr>
<tr>
<td></td>
<td>argv[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[..] (pointer)</td>
<td>4 * x</td>
</tr>
<tr>
<td></td>
<td>argv[n - 1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[n] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>envp[0] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[..] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[term] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>auxv[0] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[1] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[..] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[term] (Elf32_auxv_t)</td>
<td>8 (= AT_NULL vector)</td>
</tr>
<tr>
<td></td>
<td>padding</td>
<td>0 - 16</td>
</tr>
<tr>
<td></td>
<td>argument ASCII strings</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td></td>
<td>environment ASCII strings</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>0xbffffffffc</td>
<td>end marker</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td>0xc0000000</td>
<td>&lt; top of stack &gt;</td>
<td>0 (virtual)</td>
</tr>
</tbody>
</table>

Credit to grugq and scut
### ELF Loading

<table>
<thead>
<tr>
<th>position</th>
<th>content</th>
<th>size (bytes) + comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack pointer -&gt;</td>
<td>argc = number of args</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[0] (pointer)</td>
<td>4 (program name)</td>
</tr>
<tr>
<td></td>
<td>argv[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[..] (pointer)</td>
<td>4 * x</td>
</tr>
<tr>
<td></td>
<td>argv[n - 1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[n] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>envp[0] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[..] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[term] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>auxv[0] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[1] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[..] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[term] (Elf32_auxv_t)</td>
<td>8 (= AT_NULL vector)</td>
</tr>
<tr>
<td></td>
<td>padding</td>
<td>0 - 16</td>
</tr>
<tr>
<td></td>
<td>argument ASCIIZ strings</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td></td>
<td>environment ASCIIZ str.</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>(0xbffffffffc)</td>
<td>end marker</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td>(0xc0000000)</td>
<td>&lt; top of stack &gt;</td>
<td>0 (virtual)</td>
</tr>
</tbody>
</table>

Credit to grugq and scut
## ELF Loading

<table>
<thead>
<tr>
<th>position</th>
<th>content</th>
<th>size (bytes) + comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack pointer -&gt;</td>
<td>argc = number of args</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[0] (pointer)</td>
<td>4 (program name)</td>
</tr>
<tr>
<td></td>
<td>argv[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[..] (pointer)</td>
<td>4 * x</td>
</tr>
<tr>
<td></td>
<td>argv[n - 1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[n] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>envp[0] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[..] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[term] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>auxv[0] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[1] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[..] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[term] (Elf32_auxv_t)</td>
<td>8 (= AT_NULL vector)</td>
</tr>
<tr>
<td></td>
<td>padding</td>
<td>0 - 16</td>
</tr>
<tr>
<td></td>
<td>argument ASCIIZ strings</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td></td>
<td>environment ASCIIZ str.</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>(0xbfffffff)</td>
<td>end marker</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td>(0xc0000000)</td>
<td>&lt; top of stack &gt;</td>
<td>0 (virtual)</td>
</tr>
</tbody>
</table>

Credit to grugq and scut
## ELF Loading

<table>
<thead>
<tr>
<th>Position</th>
<th>Content</th>
<th>Size (bytes) + Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stack Pointer</td>
<td>argc = number of args</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[0] (pointer)</td>
<td>4 (program name)</td>
</tr>
<tr>
<td></td>
<td>argv[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[..] (pointer)</td>
<td>4 * x</td>
</tr>
<tr>
<td></td>
<td>argv[n - 1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>argv[n] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>envp[0] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[1] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[..] (pointer)</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>envp[term] (pointer)</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td></td>
<td>auxv[0] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[1] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[..] (Elf32_auxv_t)</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>auxv[term] (Elf32_auxv_t)</td>
<td>8 (= AT_NULL vector)</td>
</tr>
<tr>
<td></td>
<td>padding</td>
<td>0 - 16</td>
</tr>
<tr>
<td></td>
<td>argument ASCII strings</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td></td>
<td>environment ASCII strings</td>
<td>&gt;= 0</td>
</tr>
<tr>
<td>(0xbfffffffc)</td>
<td>end marker</td>
<td>4 (= NULL)</td>
</tr>
<tr>
<td>(0xc0000000)</td>
<td>top of stack</td>
<td>0 (virtual)</td>
</tr>
</tbody>
</table>

Credit to grugq and scut
# ELF Loading

<table>
<thead>
<tr>
<th>position</th>
<th>content</th>
<th>size (bytes) + comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>stack pointer -&gt;</td>
<td>[ argc = number of args ] 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ argv[0] (pointer) ] 4</td>
<td>(program name)</td>
</tr>
<tr>
<td></td>
<td>[ argv[1] (pointer) ] 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ argv[..] (pointer) ] 4 * x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ argv[n - 1] (pointer) ] 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ argv[n] (pointer) ] 4 (= NULL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ envp[0] (pointer) ] 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ envp[1] (pointer) ] 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ envp[..] (pointer) ] 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ envp[term] (pointer) ] 4 (= NULL)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ auxv[0] (Elf32_auxv_t) ] 8</td>
<td>(= AT_NULL vector)</td>
</tr>
<tr>
<td></td>
<td>[ auxv[1] (Elf32_auxv_t) ] 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ auxv[..] (Elf32_auxv_t) ] 8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ auxv[term] (Elf32_auxv_t) ] 8 (= AT_NULL vector)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ padding ] 0 - 16</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ argument ASCIIZ strings ] &gt;= 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>[ environment ASCIIZ str. ] &gt;= 0</td>
<td></td>
</tr>
<tr>
<td>(0xbfffffffc)</td>
<td>[ end marker ] 4 (= NULL)</td>
<td></td>
</tr>
<tr>
<td>(0xc0000000)</td>
<td>&lt; top of stack &gt; 0 (virtual)</td>
<td></td>
</tr>
</tbody>
</table>

Credit to grugq and scut
## ELF Auxiliary Vectors

**LD_SHOW_AUXV=1 ./main**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_SYSINFO_EHDR</td>
<td>0x7ffd9e178000</td>
</tr>
<tr>
<td>AT_MINSIGSTKSZ</td>
<td>1440</td>
</tr>
<tr>
<td>AT_HWCAP</td>
<td>b7ebfbff</td>
</tr>
<tr>
<td>AT_PAGESZ</td>
<td>4096</td>
</tr>
<tr>
<td>AT_CLKTCK</td>
<td>100</td>
</tr>
<tr>
<td>AT_PHDR</td>
<td>0x557a0890f040</td>
</tr>
<tr>
<td>AT_PHENT</td>
<td>56</td>
</tr>
<tr>
<td>AT_PHNUM</td>
<td>13</td>
</tr>
<tr>
<td>AT_BASE</td>
<td>0x7fafa043f000</td>
</tr>
<tr>
<td>AT_FLAGS</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_ENTRY</td>
<td>0x557a08910080</td>
</tr>
<tr>
<td>AT_UID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EUID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_GID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EGID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_SECURE</td>
<td>0</td>
</tr>
<tr>
<td>AT_RANDOM</td>
<td>0x7ffd9e166599</td>
</tr>
<tr>
<td>AT_HWCAP2</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_EXECFN</td>
<td>./main_argv</td>
</tr>
<tr>
<td>AT_PLATFORM</td>
<td>x86_64</td>
</tr>
</tbody>
</table>
ELF Auxiliary Vectors

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_SYSINFO_EHDR</td>
<td>0x7ffd9e178000</td>
</tr>
<tr>
<td>AT_MINSIGSTKSZ</td>
<td>1440</td>
</tr>
<tr>
<td>AT_HWCAP</td>
<td>b7ebfbff</td>
</tr>
<tr>
<td>AT_PAGESZ</td>
<td>4096</td>
</tr>
<tr>
<td>AT_CLKTCK</td>
<td>100</td>
</tr>
<tr>
<td>AT_PHDR</td>
<td>0x557a0890f040</td>
</tr>
<tr>
<td>AT_PHENT</td>
<td>56</td>
</tr>
<tr>
<td>AT_PHNUM</td>
<td>13</td>
</tr>
<tr>
<td>AT_BASE</td>
<td>0x7fafa043f000</td>
</tr>
<tr>
<td>AT_FLAGS</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_ENTRY</td>
<td>0x557a08910080</td>
</tr>
<tr>
<td>AT_UID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EUID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_GID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EGID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_SECURE</td>
<td>0</td>
</tr>
<tr>
<td>AT_RANDOM</td>
<td>0x7ffd9e166599</td>
</tr>
<tr>
<td>AT_HWCAP2</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_EXECFN</td>
<td>./main_argv</td>
</tr>
<tr>
<td>AT_PLATFORM</td>
<td>x86_64</td>
</tr>
</tbody>
</table>
ELF Auxiliary Vectors

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_SYSINFO_EHDR</td>
<td>0x7ffd9e178000</td>
</tr>
<tr>
<td>AT_MINSIGSTKSZ</td>
<td>1440</td>
</tr>
<tr>
<td>AT_HWCAP</td>
<td>b7ebfbff</td>
</tr>
<tr>
<td>AT_PAGESZ</td>
<td>4096</td>
</tr>
<tr>
<td>AT_CLKTCK</td>
<td>100</td>
</tr>
<tr>
<td>AT_PHDR</td>
<td>0x557a0890f040</td>
</tr>
<tr>
<td>AT_PHENT</td>
<td>56</td>
</tr>
<tr>
<td>AT_PHNUM</td>
<td>13</td>
</tr>
<tr>
<td>AT_BASE</td>
<td>0x7fafa043f000</td>
</tr>
<tr>
<td>AT_FLAGS</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_ENTRY</td>
<td>0x557a08910080</td>
</tr>
<tr>
<td>AT_UID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EUID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_GID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EGID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_SECURE</td>
<td>0</td>
</tr>
<tr>
<td>AT_RANDOM</td>
<td>0x7ffd9e166599</td>
</tr>
<tr>
<td>AT_HWCAP2</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_EXECFN</td>
<td>./main_argv</td>
</tr>
<tr>
<td>AT_PLATFORM</td>
<td>x86_64</td>
</tr>
</tbody>
</table>

Pointer to 16 random bytes used to implement a stack canary.
ELF Auxiliary Vectors

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT_SYSINFO_EHDR</td>
<td>0x7fffd9e178000</td>
</tr>
<tr>
<td>AT_MINSIGSTKSZ</td>
<td>1440</td>
</tr>
<tr>
<td>AT_HWCAP</td>
<td>b7ebfbff</td>
</tr>
<tr>
<td>AT_PAGESZ</td>
<td>4096</td>
</tr>
<tr>
<td>AT_CLKTCK</td>
<td>100</td>
</tr>
<tr>
<td>AT_PHDR</td>
<td>0x557a0890f040</td>
</tr>
<tr>
<td>AT_PHENT</td>
<td>56</td>
</tr>
<tr>
<td>AT_PHNUM</td>
<td>13</td>
</tr>
<tr>
<td>AT_BASE</td>
<td>0x7fafa043f000</td>
</tr>
<tr>
<td>AT_FLAGS</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_ENTRY</td>
<td>0x557a08910080</td>
</tr>
<tr>
<td>AT_UID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EUID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_GID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_EGID</td>
<td>1000</td>
</tr>
<tr>
<td>AT_SECURE</td>
<td>0</td>
</tr>
<tr>
<td>AT_RANDOM</td>
<td>0x7fffd9e166599</td>
</tr>
<tr>
<td>AT_HWCAP2</td>
<td>0x0</td>
</tr>
<tr>
<td>AT_EXECFN</td>
<td>./main_argv</td>
</tr>
<tr>
<td>AT_PLATFORM</td>
<td>x86_64</td>
</tr>
</tbody>
</table>

Pointer to page containing the Virtual Dynamic Shared Object (VDSO)
Virtual Dynamic Shared Object (VDSO)

In the old days, system calls were made using interrupts.

```
mov eax, 1    ; exit system call
mov ebx, 42   ; argument
int 0x80
```

And they were slow...
Virtual Dynamic Shared Object (VDSO)

Intel: We can do better. Thus, sysenter/sysexit

AMD: We can do better. Thus, syscall/sysret

In 32-bit mode, ‘int 0x80’ works, sometimes ‘sysenter’ works, and sometimes ‘syscall’ works.
Virtual Dynamic Shared Object (VDSO)

Linux devs make VDSO

- Dynamic Library
- Linked to all executables
- Provides __kernel_vsycall function that chooses among the variations
- Used by glibc

Again, VDSO is found in the ELF Auxiliary Vectors
Virtual Dynamic Shared Object (VDSO)

A kernel developer realized it would be useful to put other functions in VDSO (e.g. \_vdso\_clock\_gettime)
NAME

vdsO - overview of the virtual ELF dynamic shared object

SYNOPSIS

#include <sys/auxv.h>

void *vdsO = (uintptr_t) getauxval(AT_SYSINFO_EHDR);

DESCRIPTION

The "vDSO" (virtual dynamic shared object) is a small shared library that the kernel automatically maps into the address space of all user-space applications. Applications usually do not need to concern themselves with these details as the vDSO is most commonly called by the C library. This way you can code in the normal way using standard functions and the C library will take care of the rest.

__vdsO_clock_gettime LINUX_2.6
__vdsO_getcpu LINUX_2.6
__vdsO_gettimeofday LINUX_2.6
## i386 functions
The table below lists the symbols exported by the vDSO.

<table>
<thead>
<tr>
<th>symbol</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>__kernel_sigreturn</td>
<td>LINUX_2.5</td>
</tr>
<tr>
<td>__kernel_rt_sigreturn</td>
<td>LINUX_2.5</td>
</tr>
<tr>
<td>__kernel_vsyscall</td>
<td>LINUX_2.5</td>
</tr>
<tr>
<td>__vdso_clock_gettime</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_gettimeofday</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_time</td>
<td>LINUX_2.6</td>
</tr>
</tbody>
</table>

## x86-64 functions
The table below lists the symbols exported by the vDSO. All of these symbols are also available without the "__vdso_" prefix, but you should ignore those and stick to the names below.

<table>
<thead>
<tr>
<th>symbol</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>__vdso_clock_gettime</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_getcpu</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_gettimeofday</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_time</td>
<td>LINUX_2.6</td>
</tr>
</tbody>
</table>
### i386 functions

The table below lists the symbols exported by the vDSO.

<table>
<thead>
<tr>
<th>symbol</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>__kernel_sigreturn</td>
<td>LINUX_2.5</td>
</tr>
<tr>
<td>__kernel_rt_sigreturn</td>
<td>LINUX_2.5</td>
</tr>
<tr>
<td>__kernel_vsyscall</td>
<td>LINUX_2.5</td>
</tr>
<tr>
<td>__vdso_clock_gettime</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_gettimeofday</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_time</td>
<td>LINUX_2.6</td>
</tr>
</tbody>
</table>

*Note: __vdso_clock_gettime* is exported since Linux 3.15.

### x86-64 functions

The table below lists the symbols exported by the vDSO. All of these symbols are also available without the "__vdso_" prefix, but you should ignore those and stick to the names below.

<table>
<thead>
<tr>
<th>symbol</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>__vdso_clock_gettime</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_getcpu</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_gettimeofday</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_time</td>
<td>LINUX_2.6</td>
</tr>
</tbody>
</table>
__vdso_time    LINUX_2.6 (exported since Linux 3.15)

**x86-64 functions**
The table below lists the symbols exported by the vDSO. All of these symbols are also available without the "__vdso_" prefix, but you should ignore those and stick to the names below.

<table>
<thead>
<tr>
<th>symbol</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>__vdso_clock_gettime</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_getcpu</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_gettimeofday</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_time</td>
<td>LINUX_2.6</td>
</tr>
</tbody>
</table>

**x86/x32 functions**
The table below lists the symbols exported by the vDSO.

<table>
<thead>
<tr>
<th>symbol</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>__vdso_clock_gettime</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>__vdso_getcpu</td>
<td>LINUX_2.6</td>
</tr>
<tr>
<td>vdso_gettimeofday</td>
<td>LINUX_2.6</td>
</tr>
</tbody>
</table>
x86_64 syscalls

stdout equ 1 ; Constants
SYS_WRITE equ 1
SYS_EXIT equ 60
.data
string db 10,"Hello, world!",10
.code
_start:
    mov edx, sizeof string
    mov rsi, offset string
    mov edi, stdout
    mov eax, SYS_WRITE
    syscall
    mov eax, SYS_EXIT
    syscall
end _start
x86_64 syscalls

```assembly
.stdout equ 1 ; Constants
SYS_WRITE equ 1
SYS_EXIT equ 60

.data ; Data
.string db 10,"Hello, world!",10

.code ; Assembly
_start:
    mov edx, sizeof string
    mov rsi, offset string
    mov edi, stdout
    mov eax, SYS_WRITE
    syscall
    mov eax, SYS_EXIT
    syscall
end _start
```
x86_64 syscalls

stdout equ 1 ; Constants
SYS_WRITE equ 1
SYS_EXIT equ 60

.data ; Data
string db 10,"Hello, world!",10

.code ; Assembly
_start:                     
mov edx, sizeof string
mov rsi, offset string
mov edi, stdout
mov eax, SYS_WRITE
syscall
mov eax, SYS_EXIT
syscall
end _start
x86_64 syscalls

```
stdout equ 1 ; Constants
SYS_WRITE equ 1
SYS_EXIT equ 60
.data
string db 10,"Hello, world!",10
.code
_start:
    mov edx, sizeof string
    mov rsi, offset string
    mov edi, stdout
    mov eax, SYS_WRITE
    syscall
    mov eax, SYS_EXIT
    syscall
    end _start
```
x86_64 syscalls

stdout equ 1 ; Constants
SYS_WRITE equ 1
SYS_EXIT equ 60

.data
string db 10,"Hello, world!",10

.code
_start:
    mov edx, sizeof string
    mov rsi, offset string
    mov edi, stdout
    mov eax, SYS_WRITE
    syscall
    mov eax, SYS_EXIT
    syscall
    end _start
Syscalls in Linux

- Prefer using glibc
- Failing that, use `syscall` in glibc
- Failing that, use nolibc
[D]eveloping lock-free code may be cool but trying to find bugs in it is most definitely not.
—Fedor Pikus
[D]eveloping lock-free code may be cool but trying to find bugs in it is most definitely not.
—Fedor Pikus
A look at Atomics

```c
void set(int & value, bool & is_ready) {
    value = 42;
    is_ready = true;
}
```
A look at Atomics

void set( int & value, bool & is_ready ) {
    value = 42;
    is_ready = true;
}

set(int&, bool&):
    mov    dword ptr [rdi], 42
    mov    byte ptr [rsi], 1
    ret
A look at Atomics

void set( int & value, bool & is_ready ) {
    value = 42;
    is_ready = true;
}

void set_atomic( int & value, std::atomic<bool> & is_ready ) {
    value = 42;
    is_ready.store(true, std::memory_order_release);
}

set(int&, bool&):
    mov    dword ptr [rdi], 42
    mov    byte ptr [rsi], 1
    ret
A look at Atomics

void set(int & value, bool & is_ready) {
    value = 42;
    is_ready = true;
}

void set_atomic(int & value, std::atomic<bool> & is_ready) {
    value = 42;
    is_ready.store(true, std::memory_order_release);
}

set(int&, bool&):
    mov    dword ptr [rdi], 42
    mov    byte ptr [rsi], 1
    ret

set_atomic(int&, std::atomic<bool>&):
    mov    dword ptr [rdi], 42
    mov    byte ptr [rsi], 1
    ret
A look at Atomics

```c
int get( int & value, bool & is_ready ) {
    if( is_ready ) {
        return value;
    } else {
        return 0;
    }
}
```
A look at Atomics

```c
int get( int & value, bool & is_ready ) {
    if( is_ready ) {
        return value;
    } else {
        return 0;
    }
}
```

```
get(int&, bool&):
    xor   eax, eax
    cmp   byte ptr [rsi], 0
    je    .LBB1_2
    mov   eax, dword ptr [rdi]
.LBB1_2:
    ret
```
A look at Atomics

```c
int get( int & value, bool & is_ready ) {
    if( is_ready ) {
        return value;
    } else {
        return 0;
    }
}

int get_atomic( int & value, std::atomic<bool> & is_ready ) {
    if( is_ready.load( std::memory_order_acquire ) ) {
        return value;
    } else {
        return 0;
    }
}
```

```assembly
get(int&, bool&):
    xor    eax, eax
    cmp    byte ptr [rsi], 0
    je .LBB1_2
    mov    eax, dword ptr [rdi]
.LBB1_2:
    ret
```
A look at Atomics

int get( int & value, bool & is_ready ) {
    if( is_ready ) {
        return value;
    } else {
        return 0;
    }
}

int get_atomic( int & value, std::atomic<bool> & is_ready ) {
    if( is_ready.load(std::memory_order_acquire) ) {
        return value;
    } else {
        return 0;
    }
}

get(int&, bool&):
    xor    eax, eax
    cmp    byte ptr [rsi], 0
    je      .LBB1_2
    mov    eax, dword ptr [rdi]
    .LBB1_2:
    ret

get_atomic(int&, std::atomic<bool>&):
    movzx   ecx, byte ptr [rsi]
    xor    eax, eax
    test   cl, 1
    je      .LBB3_2
    mov    eax, dword ptr [rdi]
    .LBB3_2:
    ret
C’mon compiler

‘mov byte ptr [rsi], 1’ is atomic
‘cmp byte ptr [rsi], 0’ is not atomic
‘movzx ecx, byte ptr [rsi]’ is atomic

- None of the “atomic” things have a LOCK annotation
- It’s strange that atomic code sometimes look the same as non-atomic code
- What’s going on here?
Maybe you should read the manual.
Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 3 (3A, 3B, 3C, & 3D):
System Programming Guide

NOTE: The Intel 64 and IA-32 Architectures Software Developer's Manual consists of four volumes: Basic Architecture, Order Number 253655; Instruction Set Reference A-Z, Order Number 325318; System Programming Guide, Order Number 325384; Model-Specific Registers, Order Number 335582. Refer to all four volumes when evaluating your design needs.

Order Number: 325384-079US
March 2023
9.1.1 Guaranteed Atomic Operations

The Intel486 processor (and newer processors since) guarantees that the following basic memory operations will always be carried out atomically:

- Reading or writing a byte.
- Reading or writing a word aligned on a 16-bit boundary.
- Reading or writing a doubleword aligned on a 32-bit boundary.

The Pentium processor (and newer processors since) guarantees that the following additional memory operations will always be carried out atomically:

- Reading or writing a quadword aligned on a 64-bit boundary.
- 16-bit accesses to uncached memory locations that fit within a 32-bit data bus.

The P6 family processors (and newer processors since) guarantee that the following additional memory operation will always be carried out atomically:

- Unaligned 16-, 32-, and 64-bit accesses to cached memory that fit within a cache line.

Processors that enumerate support for Intel® AVX (by setting the feature flag CPUID.01H:ECX.AVX[bit 28]) guarantee that the 16-byte memory operations performed by the following instructions will always be carried out atom-
Developing lock-free code may be cool but trying to find bugs in it is most definitely not.
—Fedor Pikus
Developing lock-free code may be cool but trying to find bugs in it is most definitely not.
—Fedor Pikus
Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```cpp
void aincr(std::atomic<int> & value) {
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret
Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```cpp
void aincr(std::atomic<int> & value) {
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```cpp
void aincr(std::atomic<int> & value){
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```cpp
void aincr(std::atomic<int> & value){
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret

; AArch64
.LBB0_1:
ldaxr  w8, [x0]
add    w8, w8, #1
stlxr  w9, w8, [x0]
cbnz   w9, .LBB0_1
ret
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```c
void aincr(std::atomic<int> & value)
{
    value++;
}

; Intel x86
lock add DWORD PTR [rdi], 1
ret

; AArch64
.LBB0_1:
    ldaxr w8, [x0]
    add w8, w8, #1
    stlxr w9, w8, [x0]
    cbnz w9, .LBB0_1
    ret
```
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```c
void aincr(std::atomic<int> & value){
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret

; AArch64
.LBB0_1:
    ldaxr w8, [x0]
    add w8, w8, #1
    stlxr w9, w8, [x0]
    cbnz w9, .LBB0_1
    ret
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```cpp
void aincr(std::atomic<int> & value){
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret

; AArch64
.LBB0_1:
    ldaxr  w8, [x0]
    add    w8, w8, #1
    stlxr  w9, w8, [x0]
    cbnz   w9, .LBB0_1
    ret
```
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```c
void aincr(std::atomic<int> & value){
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret

; AArch64
.LBB0_1:
    ldaxr w8, [x0]
    add w8, w8, #1
    stlxe w9, w8, [x0]
    cbnz w9, .LBB0_1
    ret
CISC and RISC

Complex Instruction Set Computer (CISC) – Intel

Reduced Instruction Set Computer (RISC) – ARM (newer Macs, Raspberry Pi, etc.)

```cpp
void aincr(std::atomic<int> & value){
    value++;
}
```

; Intel x86
lock add DWORD PTR [rdi], 1
ret

; AArch64
.LBB0_1:
    ldaxr w8, [x0]
    add w8, w8, #1
    stlxr w9, w8, [x0]
    cbnz w9, .LBB0_1
    ret
ARM v8.1-a

mov w8, #1
ldaddal w8, w8, [x0]
ret
mov w8, #1
ldaddal w8, w8, [x0]
ret
Intel and CISC

- On modern Intel CPUs there are >556 registers (Segmented Memory, SSE/AVX, debug, etc.)
- >2000 instructions
Craziest Intel Instruction

**PCMPESTRM xmm, xmm, imm8** - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string

RDX = length of second string

Imm8 is the comparison:

- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: complement result
- Bit 5: complement only valid bits
- Bit 6: bit mask or a byte mask
Craziest Intel Instruction

**PCMPESTRM xmm, xmm, imm8** - Packed Compare Explicit Length Strings, Return Mask

**RAX** = length of first string

**RDX** = length of second string

Imm8 is the comparison:

- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: compliment result
- Bit 5: compliment only valid bits
- Bit 6: bit mask or a byte mask
Craziest Intel Instruction

PCMPESTRM xmm, xmm, imm8 - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string

RDX = length of second string

Imm8 is the comparison:

- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: compliment result
- Bit 5: compliment only valid bits
- Bit 6: bit mask or a byte mask
Craziest Intel Instruction

PCMPESTRM xmm, xmm, imm8 - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string
RDX = length of second string

Imm8 is the comparison:
- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: compliment result
- Bit 5: compliment only valid bits
- Bit 6: bit mask or a byte mask
Craziest Intel Instruction

PCMPESTRM xmm, xmm, imm8 - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string

RDX = length of second string

Imm8 is the comparison:

- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: compliment result
- Bit 5: compliment only valid bits
- Bit 6: bit mask or a byte mask

<table>
<thead>
<tr>
<th>2</th>
<th>C</th>
<th>0</th>
<th>0</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>z</td>
<td>A</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Craziest Intel Instruction

PCMPESTRM xmm, xmm, imm8 - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string

RDX = length of second string

Imm8 is the comparison:

- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: complement result
- Bit 5: complement only valid bits
- Bit 6: bit mask or a byte mask
Craziest Intel Instruction

PCMPESTRM xmm, xmm, imm8 - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string
RDX = length of second string

Imm8 is the comparison:
- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
  - Bit 4: compliment result
  - Bit 5: compliment only valid bits
  - Bit 6: bit mask or a byte mask
Craziest Intel Instruction

**PCMPESTRM xmm, xmm, imm8** - Packed Compare Explicit Length Strings, Return Mask

- **RAX** = length of first string
- **RDX** = length of second string

**Imm8** is the comparison:

- Bits 0 and 1 specify data type: `uint8`, `uint16`, `int8`, `int16`
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: compliment result
- **Bit 5**: compliment only valid bits
- Bit 6: bit mask or a byte mask
Craziest Intel Instruction

PCMPESTRM xmm, xmm, imm8 - Packed Compare Explicit Length Strings, Return Mask

RAX = length of first string
RDX = length of second string

Imm8 is the comparison:
- Bits 0 and 1 specify data type: uint8, uint16, int8, int16
- Bits 2 and 3 specify test: subset, ranges, match, substring search
- Bit 4: compliment result
- Bit 5: compliment only valid bits
- Bit 6: bit mask or a byte mask
More cool things to explore

- Models for CPU-level concurrency (x86-TSO)
- Segmented Memory
- io_uring
- Return-oriented programming
- Instruction pipelining
- Microbenchmarking
Assembly Resources

- Creel’s Assembly Tutorials [https://www.youtube.com/@WhatsACreel](https://www.youtube.com/@WhatsACreel)
- Defuse online assembler [https://defuse.ca/online-x86-assembler.htm](https://defuse.ca/online-x86-assembler.htm)
- Agner Fog’s optimization page [https://www.agner.org/optimize/](https://www.agner.org/optimize/)
- Atomics in AArch64 [https://cpufun.substack.com/p/atomics-in-aarch64](https://cpufun.substack.com/p/atomics-in-aarch64)
- Register count for Intel [https://blog.yossarian.net/2020/11/30/How-many-registers-does-an-x86-64-cpu-have](https://blog.yossarian.net/2020/11/30/How-many-registers-does-an-x86-64-cpu-have)
Linux Resources

- Article on glibc/system call controversies https://lwn.net/Articles/655028/
- Assembly that launches main 
  https://github.com/torvalds/linux/blob/16a8829130ca22666ac6236178a6233208d425c3/tools/include/nolibc/arch-x86_64.h#L193
Bonus: calling main
pop rdi ; argc  (first arg, rdi)
mov rsi, rsp ; argv[] (second arg, rsi)
lea rdx, [rsi+rdi*8+1] ; then a NULL then envp (third arg, rdx)
mov environ, rdx ; save environ
xor ebp, ebp ; zero the stack frame
mov rax, rdx ; search for auxv (follows NULL after last env)
loop:
add rax, 8 ; search for auxv using rax, it follows the
cmp rbp, [rax-8] ; ... NULL after last env (rbp is zero here)
jnz loop
mov _auxv, rax ; save it into _auxv
and rsp, -16 ; x86 ABI: esp must be 16-byte aligned before call
call main ; main() returns status code, we'll exit with it.
mov edi, eax ; retrieve exit code (32 bit)
mov eax, 60 ; NR_exit == 60
syscall ; really exit
pop rdi ; argc (first arg, rdi)
mov rsi, rsp ; argv[] (second arg, rsi)

lea rdx, [rsi+rdi*8+1] ; then a NULL then envp (third arg, rdx)
mov environ, rdx ; save environ

xor ebp, ebp ; zero the stack frame
mov rax, rdx ; search for auxv (follows NULL after last env)

loop:
add rax, 8 ; search for auxv using rax, it follows the
 cmp rbp, [rax-8] ; ... NULL after last env (rbp is zero here)
jnz loop

mov _auxv, rax ; save it into _auxv
and rsp, -16 ; x86 ABI: esp must be 16-byte aligned before call
call main ; main() returns status code, we'll exit with it.
mov edi, eax ; retrieve exit code (32 bit)
mov eax, 60 ; NR_exit == 60
syscall ; really exit
pop rdi ; argc (first arg, rdi)
mov rsi, rsp ; argv[] (second arg, rsi)
lea rdx, [rsi+rdi*8+1] ; then a NULL then envp (third arg, rdx)
mov environ, rdx ; save environ
xor ebp, ebp ; zero the stack frame
mov rax, rdx ; search for auxv (follows NULL after last env)
loop:
add rax, 8 ; search for auxv using rax, it follows the
cmp rbp, [rax-8] ; ... NULL after last env (rbp is zero here)
jnz loop

mov _auxv, rax ; save it into _auxv

and rsp, -16 ; x86 ABI: esp must be 16-byte aligned before call
call main ; main() returns status code, we'll exit with it.
mov edi, eax ; retrieve exit code (32 bit)
mov eax, 60 ; NR_exit == 60
syscall ; really exit
mov rax, rax          ; search for auxv (follows NULL after last env)
loop:
  add rax, 8            ; search for auxv using rax, it follows the
  cmp rbp, [rax-8]      ; ... NULL after last env (rbp is zero here)
  jnz loop
  mov __auxv, rax       ; save it into __auxv
  and rsp, -16          ; x86 ABI: esp must be 16-byte aligned before call
  call main            ; main() returns status code, we'll exit with it.
  mov edi, eax          ; retrieve exit code (32 bit)
  mov eax, 60           ; NR_exit == 60
  syscall              ; really exit
  hlt                   ; ensure it does not return
mov rax, rax
loop:
add rax, 8 ; search for auxv using rax, it follows the
cmp rbp, [rax-8] ; ... NULL after last env (rbp is zero here)
jnz loop
mov _auxv, rax ; save it into _auxv
and rsp, -16 ; x86 ABI: esp must be 16-byte aligned before call
call main ; main() returns status code, we'll exit with it.
mov edi, eax ; retrieve exit code (32 bit)
mov eax, 60 ; NR_exit == 60
syscall ; really exit
hlt ; ensure it does not return
mov rax, rax
loop:
add rax, 8
    ; search for auxv using rax, it follows the
cmp rbp, [rax-8]
    ; ... NULL after last env (rbp is zero here)
jnz loop
mov _auxv, rax
    ; save it into _auxv
and rsp, -16
    ; x86 ABI: esp must be 16-byte aligned before call
call main
    ; main() returns status code, we'll exit with it.
mov edi, eax
    ; retrieve exit code (32 bit)
mov eax, 60
    ; NR_exit == 60
syscall
    ; really exit
hlt
    ; ensure it does not return
mov rax, rax
loop:
    add rax, 8
    ; search for auxv using rax, it follows the
    cmp rbp, [rax-8]
    ; ... NULL after last env (rbp is zero here)
    jnz loop

    mov _auxv, rax
    ; save it into _auxv
    and rsp, -16
    ; x86 ABI: esp must be 16-byte aligned before call
    call main
    ; main() returns status code, we'll exit with it.

    mov edi, eax
    ; retrieve exit code (32 bit)
    mov eax, 60
    ; NR_exit == 60
    syscall
    ; really exit

    hlt
    ; ensure it does not return
loop:
    add rax, 8               ; search for auxv using rax, it follows the
    cmp rbp, [rax-8]         ; ... NULL after last env (rbp is zero here)
    jnz loop
    mov __auxv, rax          ; save it into __auxv
    and rsp, -16             ; x86 ABI: esp must be 16-byte aligned before call
    call main                ; main() returns status code, we'll exit with it.
    mov edi, eax             ; retrieve exit code (32 bit)
    mov eax, 60              ; NR_exit == 60
    syscall                  ; really exit
    hlt                      ; ensure it does not return
Questions/Comments
About the artist

Dan Zucco

London-based 3D art and motion director Dan Zucco creates repeating 2D patterns and brings them to life as 3D animated loops. Inspired by architecture, music, modern art, and generative design, he often starts in Adobe Illustrator and builds his animations using Adobe After Effects and Cinema 4D. Zucco’s objective for this piece was to create a geometric design that felt like it could have an infinite number of arrangements.