ELF (1A)

Young Won Lim
3/28/16

Please send corrections (or suggestions) to youngwlim@hotmail.com.

This document was produced by using OpenOffice.

Young Won Lim
3/28/16

ELF

Each ELF file is made up of one ELF header,
followed by file data. The file data can include:

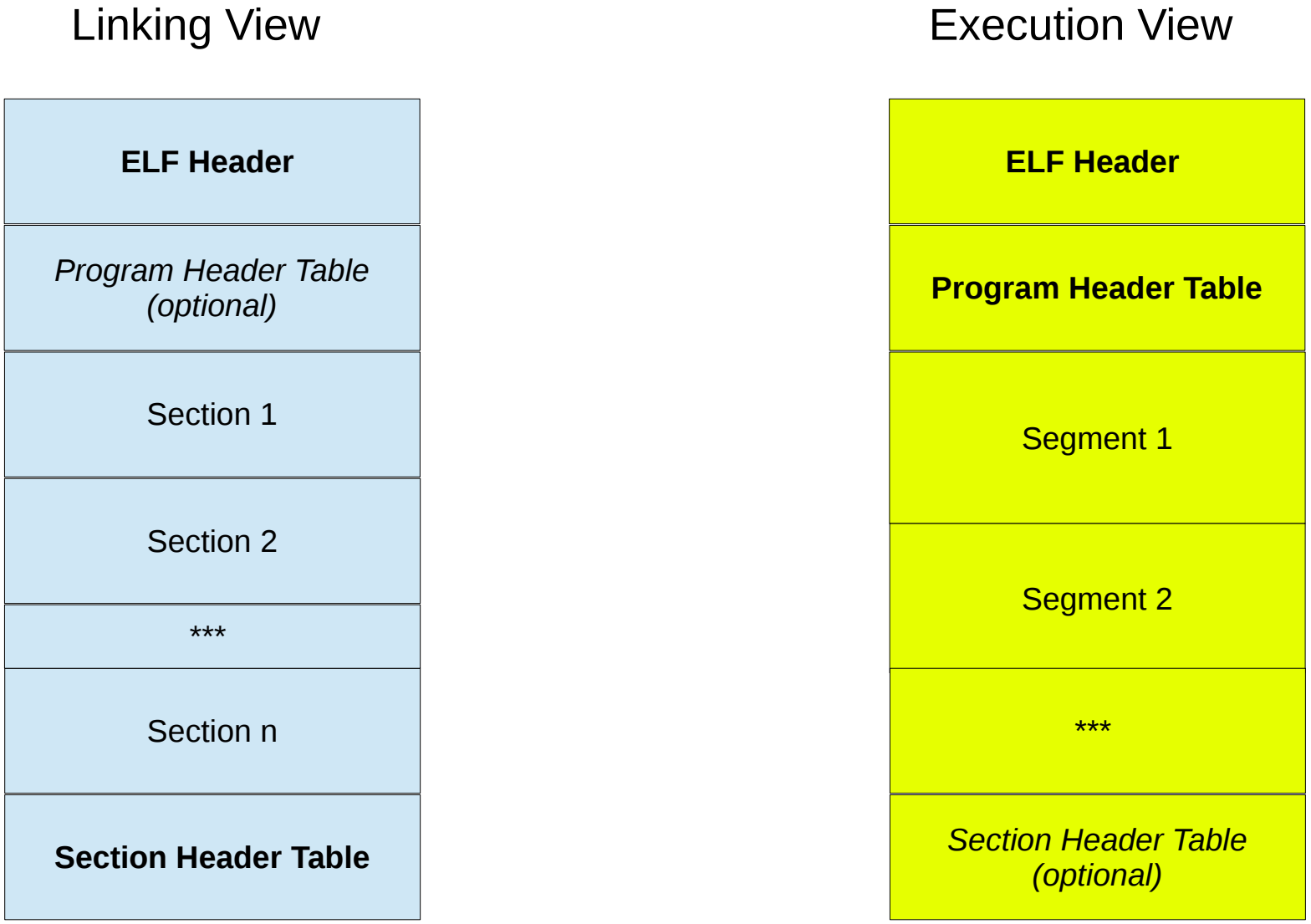
- Program header table, describing zero or more segments
- Section header table, describing zero or more sections
- Data referred to by entries in the program header table
  or section header table

The segments contain
  info about **runtime execution** of the file

The sections contain
  Info about **linking** and **relocation**

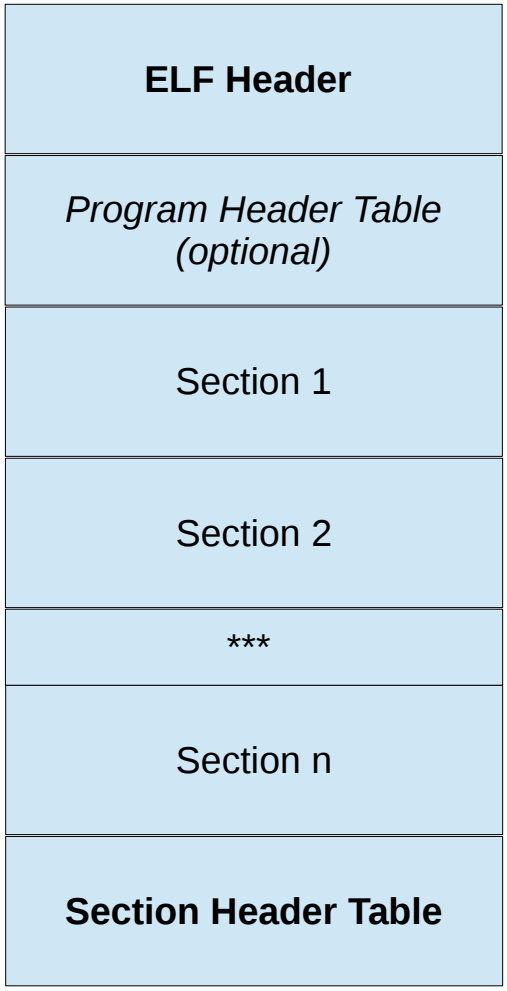
Any byte can belong to at most one section
There can be orphan bytes not owned by any section

ELF's two views



Relocatable Object Files

Linking View



.text	machine code
.rodata	read only data
.data	initialized global data
.bss	uninitialized data
.symtab	symbol table
.rel.text	
.rel.data	
.debug	
.line	
.strtab	

Tools

```
The Unix file utility can display some information about ELF files,
including the instruction set architecture
for which the code is intende
      in a  relocatable,
            executable, or
            shared object file
or on which an ELF core dump was produced.
```

objdump

readelf

elfutils

elfdump (Solaris and FreeBSD)

Example Program

// main.c

```
int a = 100;
int b = 200;

int main(void) {
      swap();
      return 0;
}
```

```
gcc -c main.c
gcc -c swap.c
gcc -o p main.o swap.o

./p
```

// swap.c

```
extern int a;
extern int b;

int *p1 = &a;
int *p2;

void swap(void) {
      int temp;

      p2 = &b;

      temp    = *p1;
      *p1     = *p2;
      *p2     = temp;
}
```

readelf -h

```
ELF Header:
  Magic:                             7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          264 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           40 (bytes)
  Number of section headers:         12
  Section header string table index: 9
```

readelf -s

```
Symbol table '.symtab' contains 12 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS main.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    6
     6: 00000000     0 SECTION LOCAL  DEFAULT    7
     7: 00000000     0 SECTION LOCAL  DEFAULT    5
     8: 00000000     4 OBJECT  GLOBAL DEFAULT    3 a
     9: 00000004     4 OBJECT  GLOBAL DEFAULT    3 b
    10: 00000000    18 FUNC    GLOBAL DEFAULT    1 main
    11: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND swap

Symbol table '.symtab' contains 13 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000     0 FILE    LOCAL  DEFAULT  ABS swap.c
     2: 00000000     0 SECTION LOCAL  DEFAULT    1
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    5
     5: 00000000     0 SECTION LOCAL  DEFAULT    7
     6: 00000000     0 SECTION LOCAL  DEFAULT    8
     7: 00000000     0 SECTION LOCAL  DEFAULT    6
     8: 00000000     4 OBJECT  GLOBAL DEFAULT    3 p1
     9: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND a
    10: 00000004     4 OBJECT  GLOBAL DEFAULT  COM p2
    11: 00000000    53 FUNC    GLOBAL DEFAULT    1 swap
    12: 00000000     0 NOTYPE  GLOBAL DEFAULT  UND b
```

// main.c

```
int a = 100;
int b = 200;

int main(void) {
      swap();
      return 0;
}
```

// swap.c

```
extern int a;
extern int b;

int *p1 = &a;
int *p2;

void swap(void) {
      int temp;

      p2 = &b;

      temp    = *p1;
      *p1     = *p2;
      *p2     = temp;
}
```

readelf -S

```
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000012 00  AX  0   0  1
  [ 2] .rel.text         REL             00000000 0003c0 000008 08      10   1  4
  [ 3] .data             PROGBITS        00000000 000048 000008 00  WA  0   0  4
  [ 4] .bss              NOBITS          00000000 000050 000000 00  WA  0   0  1
  [ 5] .comment          PROGBITS        00000000 000050 000025 01  MS  0   0  1
  [ 6] .note.GNU-stack   PROGBITS        00000000 000075 000000 00      0   0  1
  [ 7] .eh_frame         PROGBITS        00000000 000078 000038 00   A  0   0  4
  [ 8] .rel.eh_frame     REL             00000000 0003c8 000008 08      10   7  4
  [ 9] .shstrtab         STRTAB          00000000 0000b0 000057 00      0   0  1
  [10] .symtab           SYMTAB          00000000 0002e8 0000c0 10      11   8  4
  [11] .strtab           STRTAB          00000000 0003a8 000016 00      0   0  1
Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text             PROGBITS        00000000 000034 000035 00  AX  0   0  1
  [ 2] .rel.text         REL             00000000 00041c 000030 08      11   1  4
  [ 3] .data             PROGBITS        00000000 00006c 000004 00  WA  0   0  4
  [ 4] .rel.data         REL             00000000 00044c 000008 08      11   3  4
  [ 5] .bss              NOBITS          00000000 000070 000000 00  WA  0   0  1
  [ 6] .comment          PROGBITS        00000000 000070 000025 01  MS  0   0  1
  [ 7] .note.GNU-stack   PROGBITS        00000000 000095 000000 00      0   0  1
  [ 8] .eh_frame         PROGBITS        00000000 000098 000038 00   A  0   0  4
  [ 9] .rel.eh_frame     REL             00000000 000454 000008 08      11   8  4
  [10] .shstrtab         STRTAB          00000000 0000d0 00005b 00      0   0  1
  [11] .symtab           SYMTAB          00000000 000334 0000d0 10      12   8  4
  [12] .strtab           STRTAB          00000000 000404 000017 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)
```

readelf -l

```
Elf file type is EXEC (Executable file)
Entry point 0x80482f0
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00120 0x00120 R E 0x4
  INTERP         0x000154 0x08048154 0x08048154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x005d4 0x005d4 R E 0x1000
  LOAD           0x000f08 0x08049f08 0x08049f08 0x00120 0x00128 RW  0x1000
  DYNAMIC        0x000f14 0x08049f14 0x08049f14 0x000e8 0x000e8 RW  0x4
  NOTE           0x000168 0x08048168 0x08048168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x0004d0 0x080484d0 0x080484d0 0x00034 0x00034 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f08 0x08049f08 0x08049f08 0x000f8 0x000f8 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .gnu.hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame_hdr .eh_frame
   03     .init_array .fini_array .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
   06     .eh_frame_hdr
   07
   08     .init_array .fini_array .jcr .dynamic .got
```

tiny.c & tiny.asm

```
// tiny.c

int main(void) {
      return(42);
}
```

```
gcc -Wall tiny.c
./a.out ; echo $?
```

gcc -s : remove all symbol tables and relocation info from executables

strip : remove symbols from object files (linux executable - command)

```
gcc -Wall -s tiny.c
gcc -Wall -s -O3 tiny.c
```

```
// tiny.asm

BITS 32
GLOBAL main
SECTION .text
main:
      mov     eax,     42
      ret
```

```
nasm -f elf tiny.asm
gcc -Wall -s tiny.o
./a.out ; echo $?
```

```
gcc -Wall -s -nostartfiles tiny.o
```

Start files & Standard Libraries

-nostartfiles

Do not use the standard system startup files when linking.

The standard system libraries are used normally, unless -nostdlib or -nodefaultlibs is used.

-nodefaultlibs

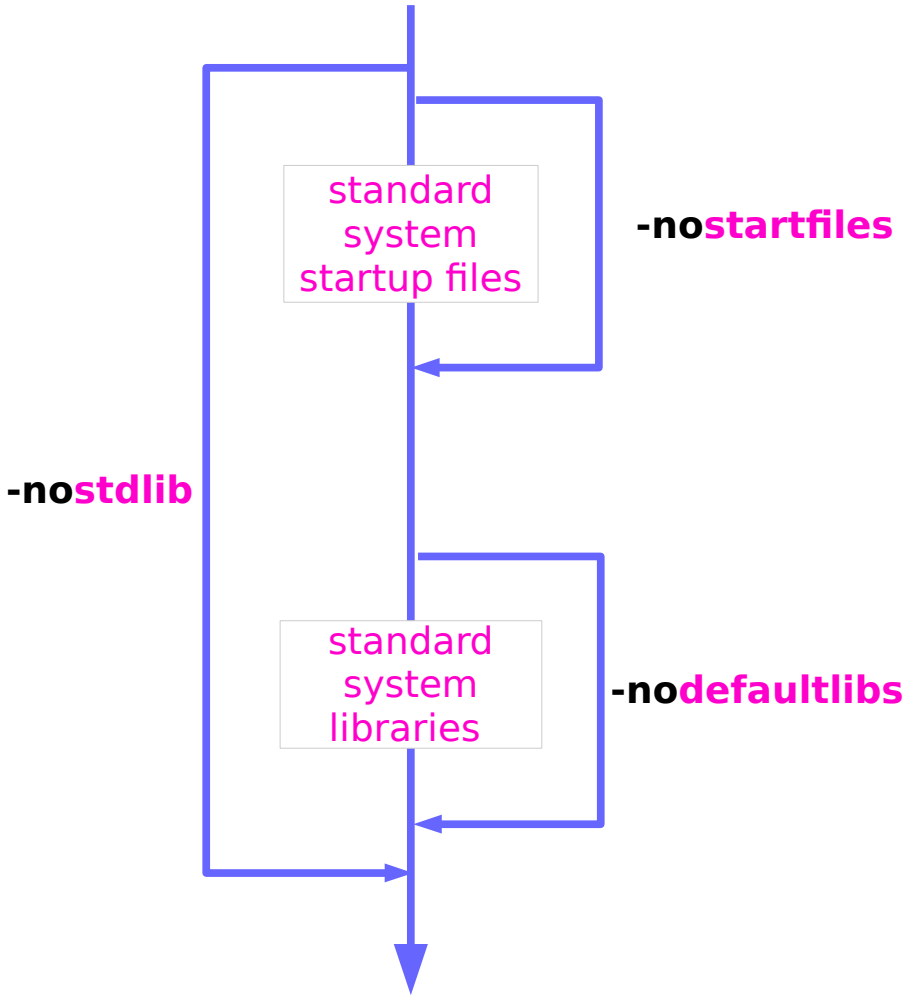
Do not use the standard system libraries when linking.

Only the libraries you specify will be passed to the linker.

The standard startup files are used normally, unless -nostartfiles is used.

-nostdlib

Do not use the standard system startup files or libraries when linking.



http://linux.die.net/man/1/gcc

libc.a & libgcc.a

The compiler may generate calls to *memcmp*, *memset*, *memcpy* and *memmove*. These entries are usually resolved by entries in **libc**. These entry points should be supplied through some other mechanism when this option is specified.

One of the standard libraries bypassed by **-nostdlib** and **-nodefaultlibs** is **libgcc.a**, a library of internal subroutines that GCC uses to overcome shortcomings of particular machines, or special needs for some languages.

In most cases, you need **libgcc.a** even when you want to avoid other standard libraries. In other words, when you specify **-nostdlib** and **-nodefaultlibs** you should usually specify **-lgcc** as well. This ensures that you have no unresolved references to internal GCC library subroutines. (For example, **__main**, used to ensure C++ constructors will be called.)

libc

- *memcmp()*
- *memset()*
- *memcpy()*
- *memmove()*

http://linux.die.net/man/1/gcc

tiny.c results

```
young@USB01:~$ cat tiny.c
int main(void) { return 42; }

young@USB01:~$ gcc -Wall tiny.c
young@USB01:~$ ./a.out
young@USB01:~$ echo $?
42
young@USB01:~$ wc -c a.out
8500 a.out
young@USB01:~$
young@USB01:~$ gcc -Wall -s tiny.c
young@USB01:~$ wc -c a.out
6272 a.out
young@USB01:~$
young@USB01:~$ gcc -Wall -s -O3 tiny.c
young@USB01:~$ wc -c a.out
6272 a.out
young@USB01:~$
```

tiny.asm results

```
young@USB01:~$ cat tiny.asm
BITS 64
GLOBAL main
SECTION .text
main:
        mov eax, 42
        ret

young@USB01:~$
young@USB01:~$ nasm -f elf64 tiny.asm
young@USB01:~$ gcc -Wall -s tiny.o
young@USB01:~$ ./a.out
young@USB01:~$ echo $?
42
young@USB01:~$ wc -c a.out
6272 a.out
young@USB01:~$
young@USB01:~$ gcc -Wall -s -nostartfiles tiny.o
/usr/bin/ld: warning: cannot find entry symbol _start; defaulting to 000000000040
0240
young@USB01:~$ wc -c a.out
4832 a.out
young@USB01:~$ ./a.out
Segmentation fault (core dumped)
young@USB01:~$
```

Calling external _exit()

```
// tiny.asm

BITS 32
GLOBAL main
SECTION .text
main:
      mov     eax,     42
      ret
```

```
nasm -f elf tiny.asm
gcc -Wall -s tiny.o
./a.out ; echo $?
```

```
// tiny.asm

BITS 32
GLOBAL _start
EXTERN _exit
SECTION .text
_start:
      push    dword   42
      call      _exit
```

```
nasm -f elf tiny.asm
gcc -Wall -s -nostartfiles tiny.o
./a.out ; echo $?
```

Calling external _exit() : results

```
young@USB01:~$ cat tiny.asm
BITS 64
GLOBAL _start
EXTERN _exit
SECTION .text
_start:
        mov rdi, 42  ; rdi hold 1st argument
        call _exit

young@USB01:~$
young@USB01:~$ nasm -f elf64 tiny.asm
young@USB01:~$
young@USB01:~$ gcc -Wall -s -nostartfiles tiny.o
young@USB01:~$
young@USB01:~$ ./a.out
young@USB01:~$ echo $?
42
young@USB01:~$ wc -c a.out
5224 a.out
young@USB01:~$
```

Calling System Call exit()

```
// tiny.asm

BITS 32
GLOBAL _start
SECTION .text
_start:
    mov     eax,    1
    mov     ebx,    42
    int     0x80
```

System call number 1: exit()

Argument : 42

Passes control to interrupt vector
invokes system call

```
nasm -f elf tiny.asm
gcc -Wall -s -nostdlib tiny.o
./a.out ; echo $?
```

Software Interrupt INT

INT (x86 instruction)

From Wikipedia, the free encyclopedia

INT is an assembly language instruction for x86 processors that generates a software interrupt. It takes the interrupt number formatted as a byte value.[1]

When written in assembly language, the instruction is written like this:

```
INT X
```

where `X` is the software interrupt that should be generated (0-255).

Depending on the context, compiler, or assembler, a software interrupt number is often given as a hexadecimal value, sometimes with a prefix *0x* or the suffix *h*. For example, `INT 21H` will generate the software interrupt 0x21 (33 in decimal), causing the function pointed to by the 34th vector in the interrupt table to be executed, which is typically an MS-DOS API call.

Most Unix systems and derivatives do not use software interrupts, with the exception of interrupt 0x80, used to make system calls. This is accomplished by entering a 32-bit value corresponding to a kernel function into the EAX register of the processor and then executing INT 0x80.

Calling System Call exit()

```
young@USB01:~$ cat tiny.asm
BITS 64
GLOBAL _start
SECTION .text
_start:
        mov eax, 1
        mov ebx, 42
        int 0x80


young@USB01:~$ nasm -f elf64 tiny.asm
young@USB01:~$ gcc -Wall -s -nostdlib tiny.o
young@USB01:~$ ./a.out
young@USB01:~$ echo $?
42
young@USB01:~$ wc -c a.out
528 a.out
young@USB01:~$
```

Calling System Call sys_exit()

```
young@USB01:~$ cat tiny.asm
BITS 64
GLOBAL _start
SECTION .text
_start:
        mov eax, 60
        mov edi, 42
        syscall


young@USB01:~$ nasm -f elf64 tiny.asm
young@USB01:~$ gcc -Wall -s -nostdlib tiny.o
young@USB01:~$ ./a.out
young@USB01:~$ echo $?
42
young@USB01:~$ wc -c a.out
528 a.out
young@USB01:~$
```

nasm vs. gas

```
// tiny.asm

BITS 32
GLOBAL _start
.text
_start:
      xor       eax, eax
      inc       eax
      mov       bl, 42
      int       0x80
```

nasm -f elf64 tiny.s
gcc -s -nostdlib tiny.o
./a.out ; echo $?

```
// tiny.s

.global _start
.text
_start:
      xorl      %eax, %eax
      incl      %eax
      movb      $42,      %bl
      int       $0x80
```

gcc -s -nostdlib tiny.s
./a.out ; echo $?

Hand Coding ELF files

```
BITS 32

  org  0x08048000

ehdr:             ; Elf32_Ehdr
  db  0x7F, "ELF"  ;  e_ident
  db  1, 1, 1, 0

_start:
  mov  bl, 42
  xor  eax, eax
  inc  eax
  int  0x80

  db  0
  dw  2             ;  e_type
  dw  3             ;  e_machine
  dd  1             ;  e_version
  dd  _start        ;  e_entry
  dd  phdr - $$     ;  e_phoff
  dd  0             ;  e_shoff
  dd  0             ;  e_flags
  dw  ehdrsize      ;  e_ehsize
  dw  phdrsize      ;  e_phentsize
  dw  1             ;  e_phnum
  dw  0             ;  e_shentsize
  dw  0             ;  e_shnum
  dw  0             ;  e_shstrndx

  ehdrsize  equ  $ - ehdr
```

```
phdr:             ; Elf32_Phdr
  dd  1             ;  p_type
  dd  0             ;  p_offset
  dd  $$            ;  p_vaddr
  dd  $$            ;  p_paddr
  dd  filesize      ;  p_filesz
  dd  filesize      ;  p_memsz
  dd  5             ;  p_flags
  dd  0x1000        ;  p_align

  phdrsize  equ  $ - phdr

  filesize  equ  $ - $$
```

nasm -f bin -o a.out elf1.asm
./a.out ; echo $?

```
young@USB01:~$ wc -c a.out
84 a.out
young@USB01:~$
```

Assembly for a exit() system call

```
young@USB01:~$ cat t.asm
_start:
        mov bl, 42
        xor eax, eax
        inc eax
        int 0x80

                                        0x2a = 42
young@USB01:~$ !obj
objdump -d t.o

t.o:     file format elf32-i386


Disassembly of section .text:

00000000 <_start>:
   0:   b3 2a                   mov    $0x2a,%bl
   2:   31 c0                   xor    %eax,%eax
   4:   40                      inc    %eax
   5:   cd 80                   int    $0x80
young@USB01:~$
```

Reading hand coded ELF file a.out (1)

```
young@USB01:~$ readelf -a a.out
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 b3 2a 31 c0 40 cd 80 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       179
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048008
  Start of program headers:          52 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         1
  Size of section headers:           0 (bytes)
  Number of section headers:         0
  Section header string table index: 0
```

```
0:  b3 2a                   mov    $0x2a,%bl
2:  31 c0                   xor    %eax,%eax
4:  40                      inc    %eax
5:  cd 80                   int    $0x80
```

Reading hand coded ELF file a.out (2)

```
There are no sections in this file.

There are no sections to group in this file.

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x08048000 0x08048000 0x00054 0x00054 R E 0x1000

There is no dynamic section in this file.

There are no relocations in this file.

The decoding of unwind sections for machine type Intel 80386 is not currently supported.

Dynamic symbol information is not available for displaying symbols.

No version information found in this file.
```

Nasm Tokens for address calculation

```
BITS 32

  org  0x08048000

ehdr:             ; Elf32_Ehdr
  db  0x7F, "ELF"  ;  e_ident
  db  1, 1, 1, 0

_start:
  mov  bl, 42
  xor  eax, eax
  inc  eax
  int  0x80

  db  0
  dw  2             ;  e_type
  dw  3             ;  e_machine
  dd  1             ;  e_version
  dd  _start        ;  e_entry
  dd  phdr - $$     ;  e_phoff
  dd  0             ;  e_shoff
  dd  0             ;  e_flags
  dw  ehdrsize      ;  e_ehsize
  dw  phdrsize      ;  e_phentsize
  dw  1             ;  e_phnum
  dw  0             ;  e_shentsize
  dw  0             ;  e_shnum
  dw  0             ;  e_shstrndx

  ehdrsize  equ  $ - ehdr
```

```
phdr:             ; Elf32_Phdr
  dd  1             ;  p_type
  dd  0             ;  p_offset
  dd  $$            ;  p_vaddr
  dd  $$            ;  p_paddr
  dd  filesize      ;  p_filesz
  dd  filesize      ;  p_memsz
  dd  5             ;  p_flags
  dd  0x1000        ;  p_align

  phdrsize  equ  $ - phdr

  filesize  equ  $ - $$
```

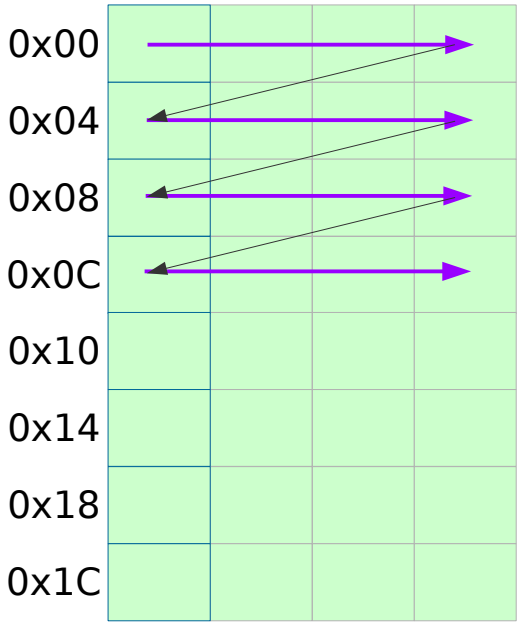
$ evaluates to the assembly position at the beginning of the line containing the expression current position

$$ evaluates to the beginning of the current section

$-$$ can tell how far into the section

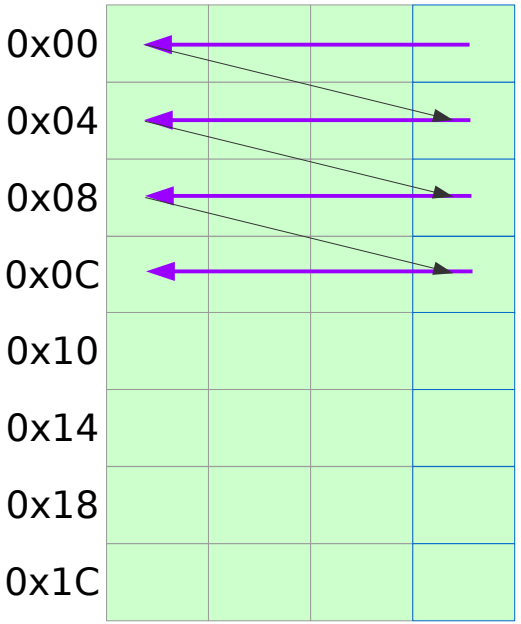
Memory Map Figures

Figure Type 1



Increasing
byte addresses

Figure Type 2



Increasing
byte addresses

Endianness

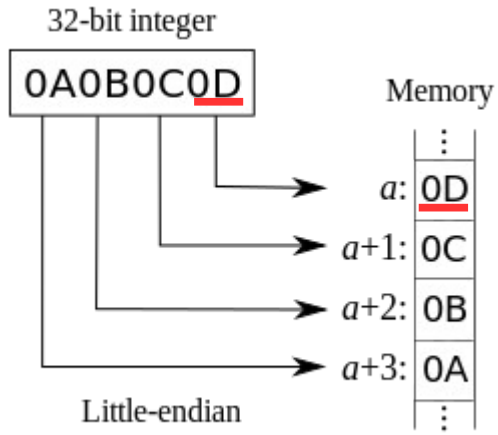
Little Endian

Figure Type 1

0x00	0D	0C	0B	0A
0x04				
0x08				
0x0C				

Figure Type 2

0x00	0A	0B	0C	0D
0x04				
0x08				
0x0C				



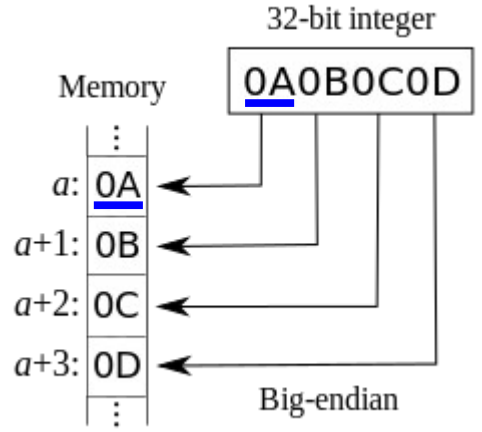
Big Endian

Figure Type 1

0x00	0A	0B	0C	0D
0x04				
0x08				
0x0C				

Figure Type 2

0x00	0D	0C	0B	0A
0x04				
0x08				
0x0C				



https://en.wikipedia.org/wiki/Endianness

Memory layout

```
BITS 32

  org  0x08048000

ehdr:             ; Elf32_Ehdr
  db  0x7F, "ELF"  ;  e_ident
  db  1, 1, 1, 0

_start:
  mov  bl, 42
  xor  eax, eax
  inc  eax
  int  0x80

  db  0
  dw  2             ;  e_type
  dw  3             ;  e_machine
  dd  1             ;  e_version
  dd  _start        ;  e_entry
  dd  phdr - $$     ;  e_phoff
  dd  0             ;  e_shoff
  dd  0             ;  e_flags
  dw  ehdrsize      ;  e_ehsize
  dw  phdrsize      ;  e_phentsize
  dw  1             ;  e_phnum
  dw  0             ;  e_shentsize
  dw  0             ;  e_shnum
  dw  0             ;  e_shstrndx

  ehdrsize  equ  $ - ehdr
```

```
phdr:             ; Elf32_Phdr
  dd  1             ;  p_type
  dd  0             ;  p_offset
  dd  $$            ;  p_vaddr
  dd  $$            ;  p_paddr
  dd  filesize      ;  p_filesz
  dd  filesize      ;  p_memsz
  dd  5             ;  p_flags
  dd  0x1000        ;  p_align

  phdrsize  equ  $ - phdr

  filesize  equ  $ - $$
```

0x34 (52)

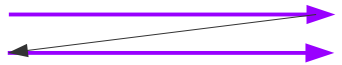
7F	'E'	'L'	'F'
1	1	1	0
b3	2a	31	c0
40	cd	80	00
2		3	
1			
00000008			
phdr - $$ = 0x34			
0			
0			
esz=0x34		psz=0x20	
1		0	
0		0	

0x20 (32)

1
0
$$ = 0x08048000
$$ = 0x08048000
filesize=0x54 (84)
filesize=0x54 (84)
5
align = 0x1000

52 + 32 = 84

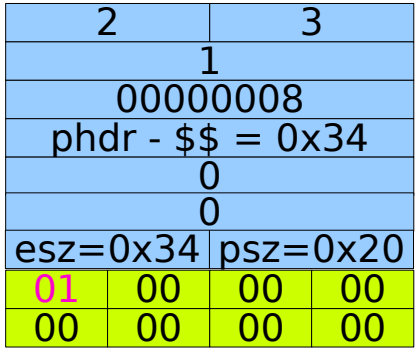
```
young@USB01:~$ wc -c a.out
84 a.out
young@USB01:~$
```



```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x08048000 0x08048000 0x00054 0x00054 R E 0x1000
```

ELF Header

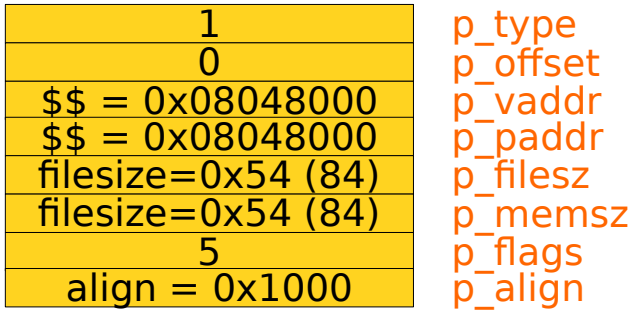
e_machine	the required architecture for an individual file.
e_version	the object file version.
e_entry	the virtual address of the entry point
e_phoff	the program header table's file offset
e_shoff	the section header table's file offset
e_flags	processor-specific flags
e_ehsize	the ELF header's size
e_phentsize	the entry size of the program header table
e_phnum	the number of entries in the program header table
e_shentsize	the entry size of a section header table
e_shnum	the number of entries in the section header table
e_shstrndx	the index to the section name string table



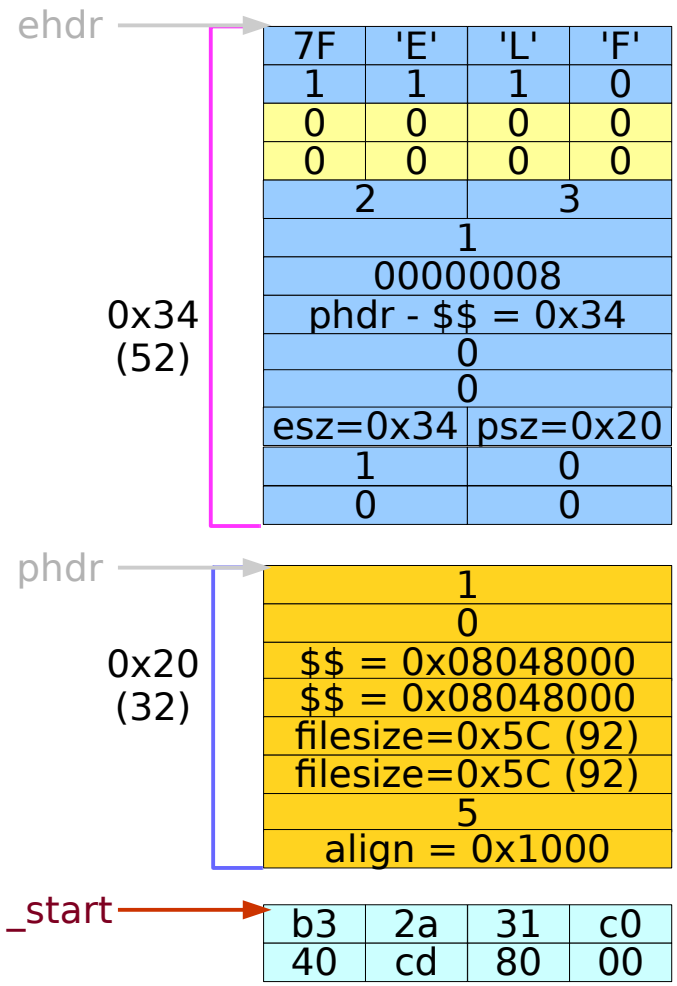
e_type, e_machine
e_version
e_entry
e_phoff
e_shoff
e_flags
e_ehsize, e_phentsize
e_phnum
e_shentsize

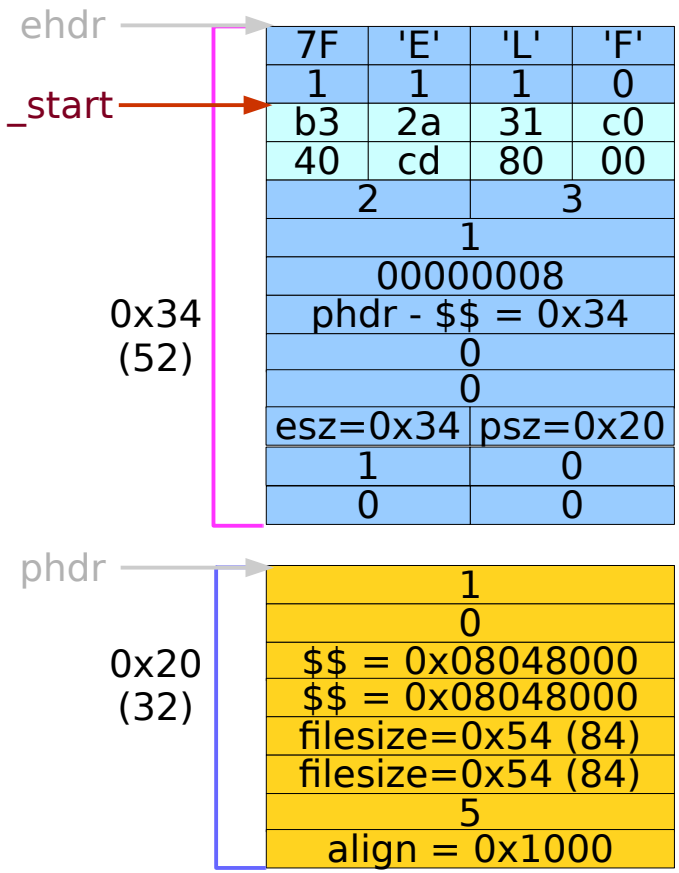
Program Header

p_type	
p_offset	the offset of the starting byte of the segment
p_vaddr	the virtual address of the starting byte of the segment in memory
p_paddr	the physical address of the starting byte of the segment in memory
p_filesz	the number of bytes in the file image of the segment
p_memsz	the number of bytes in the memory image of the segment
p_flags	flags relevant to the segment
p_align	

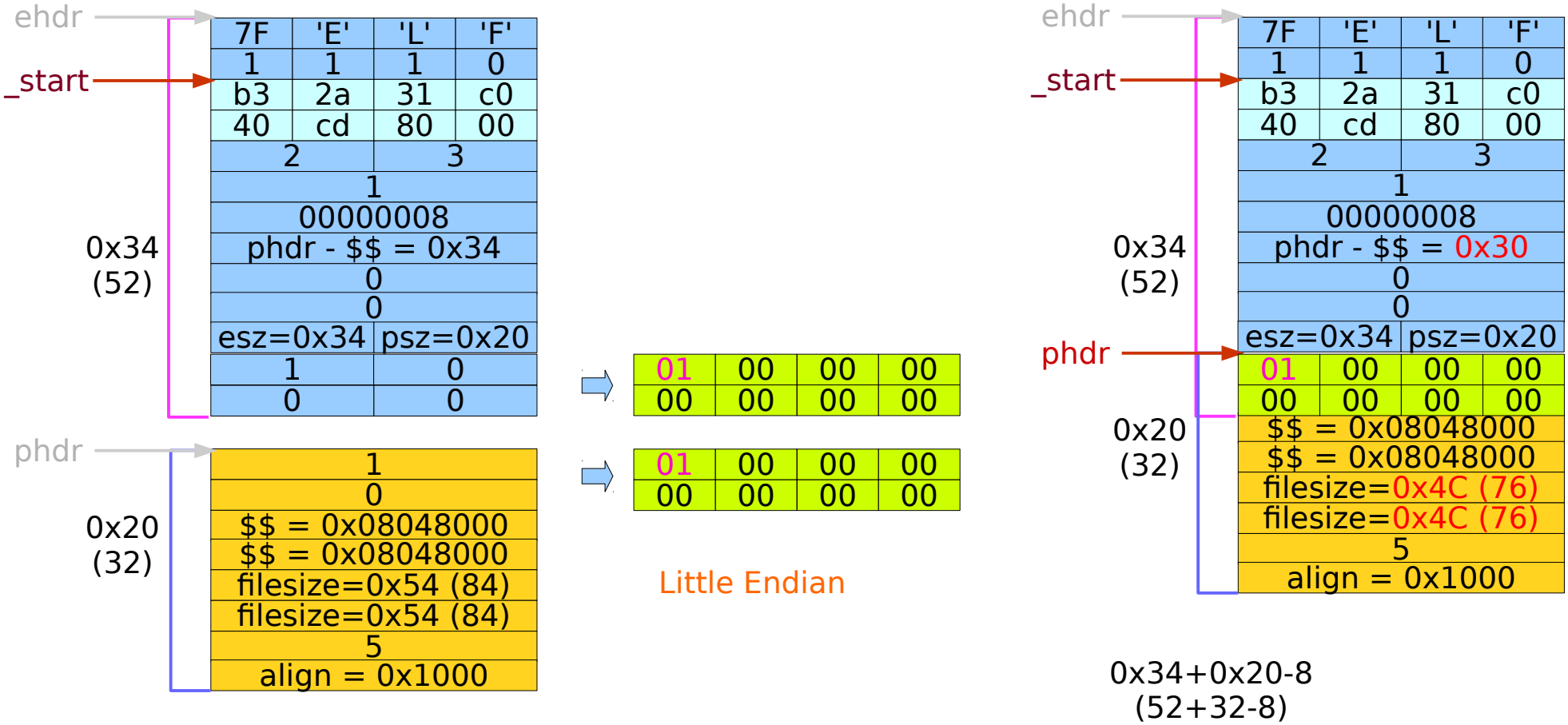


Memory layout

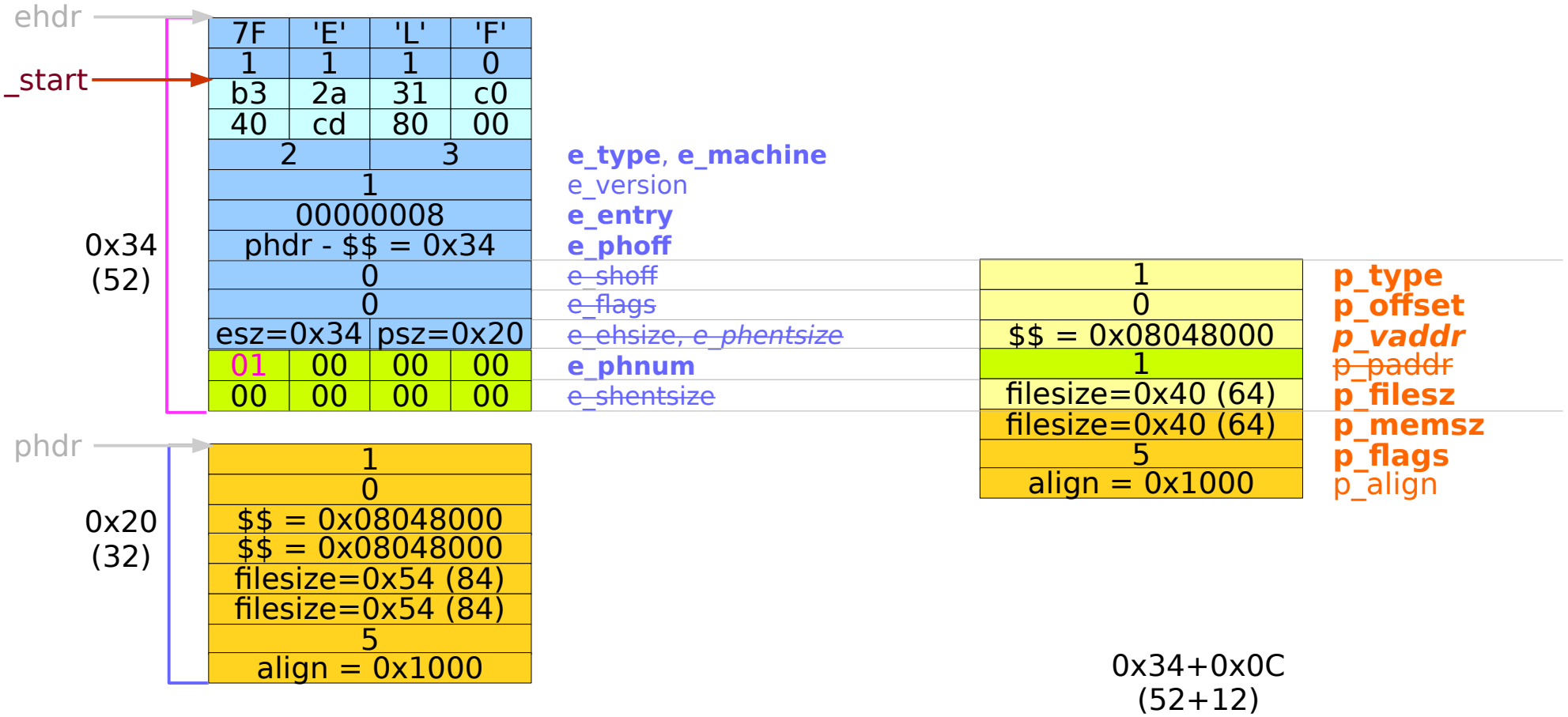




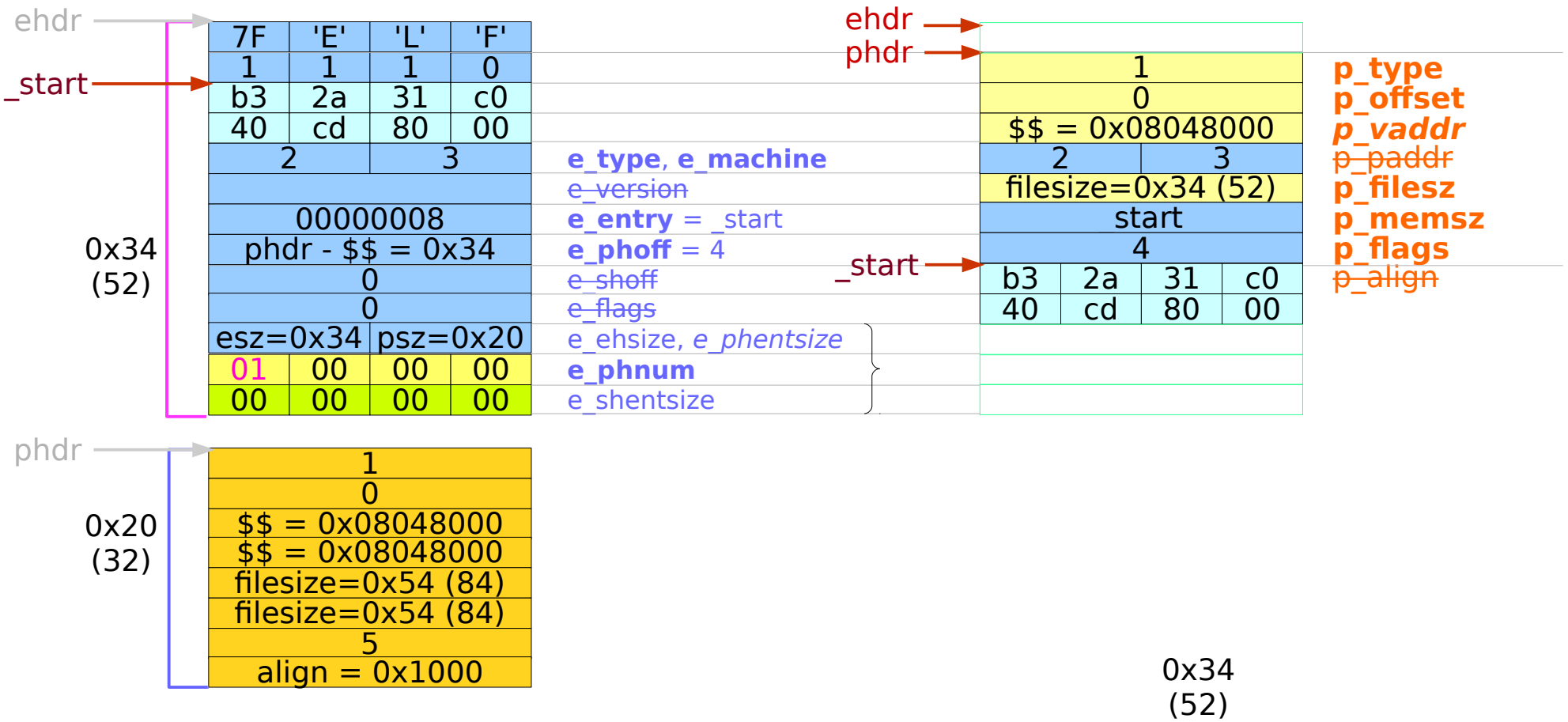
Overlay (1)


ehdr
_start
7F 'E' 'L' 'F'
1 1 1 0
b3 2a 31 c0
40 cd 80 00
2 3
1
00000008
phdr - $$ = 0x34
0
0
esz=0x34 psz=0x20
1 0
0 0
0x34
(52)
phdr
1
0
$$ = 0x08048000
$$ = 0x08048000
filesize=0x54 (84)
filesize=0x54 (84)
5
align = 0x1000
0x20
(32)
01 00 00 00
00 00 00 00
01 00 00 00
00 00 00 00
Little Endian
ehdr
_start
7F 'E' 'L' 'F'
1 1 1 0
b3 2a 31 c0
40 cd 80 00
2 3
1
00000008
phdr - $$ = 0x30
0
0
esz=0x34 psz=0x20
phdr
01 00 00 00
00 00 00 00
$$ = 0x08048000
$$ = 0x08048000
filesize=0x4C (76)
filesize=0x4C (76)
5
align = 0x1000
0x34
(52)
0x20
(32)
0x34+0x20-8
(52+32-8)

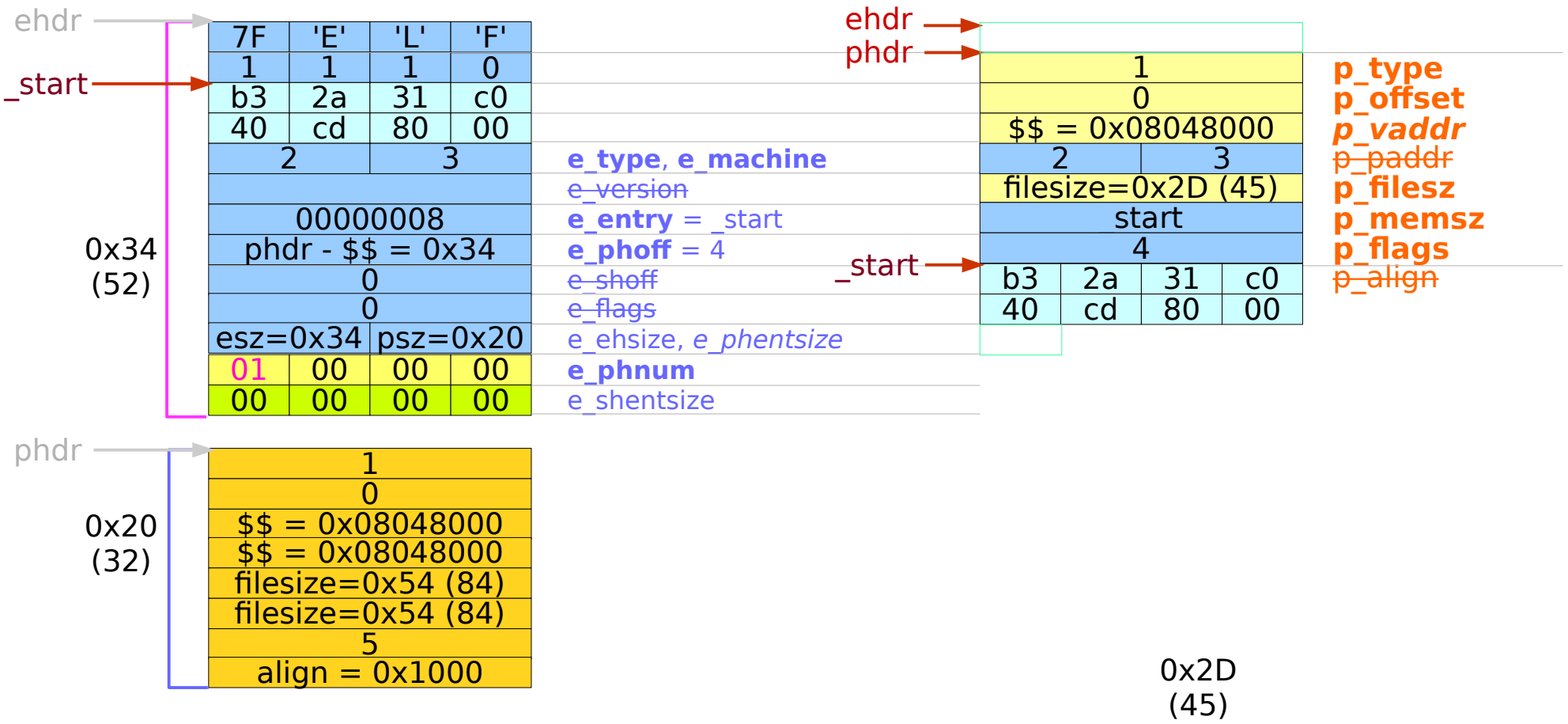
Overlay (2)


ehdr
_start
phdr
0x34
(52)
0x20
(32)
7F 'E' 'L' 'F'
1 1 1 0
b3 2a 31 c0
40 cd 80 00
2 3
1
00000008
phdr - $$ = 0x34
0
0
esz=0x34 psz=0x20
01 00 00 00
00 00 00 00
1
0
$$ = 0x08048000
$$ = 0x08048000
filesize=0x54 (84)
filesize=0x54 (84)
5
align = 0x1000
e_type, e_machine
e_version
e_entry
e_phoff
e_shoff
e_flags
e_ehsize, e_phentsize
e_phnum
e_shentsize
1
0
$$ = 0x08048000
1
filesize=0x40 (64)
filesize=0x40 (64)
5
align = 0x1000
p_type
p_offset
p_vaddr
p_paddr
p_filesz
p_memsz
p_flags
p_align
0x34+0x0C
(52+12)

Overlay (3)


ehdr
_start
7F 'E' 'L' 'F'
1 1 1 0
b3 2a 31 c0
40 cd 80 00
2 3
00000008
phdr - $$ = 0x34
0
0
esz=0x34 psz=0x20
01 00 00 00
00 00 00 00
0x34
(52)
e_type, e_machine
e_version
e_entry = _start
e_phoff = 4
e_shoff
e_flags
e_ehsize, e_phentsize
e_phnum
e_shentsize
phdr
1
0
$$ = 0x08048000
$$ = 0x08048000
filesize=0x54 (84)
filesize=0x54 (84)
5
align = 0x1000
0x20
(32)
ehdr
phdr
_start
1
0
$$ = 0x08048000
2 3
filesize=0x34 (52)
start
4
b3 2a 31 c0
40 cd 80 00
0x34
(52)
p_type
p_offset
p_vaddr
p_paddr
p_filesz
p_memsz
p_flags
p_align

Overlay (4)


ehdr
_start
7F 'E' 'L' 'F'
1 1 1 0
b3 2a 31 c0
40 cd 80 00
2 3
00000008
phdr - $$ = 0x34
0
0
esz=0x34 psz=0x20
01 00 00 00
00 00 00 00
e_type, e_machine
e_version
e_entry = _start
e_phoff = 4
e_shoff
e_flags
e_ehsize, e_phentsize
e_phnum
e_shentsize
0x34
(52)
phdr
1
0
$$ = 0x08048000
$$ = 0x08048000
filesize=0x54 (84)
filesize=0x54 (84)
5
align = 0x1000
0x20
(32)
ehdr
phdr
_start
1
0
$$ = 0x08048000
2 3
filesize=0x2D (45)
start
4
b3 2a 31 c0
40 cd 80 00
0x2D
(45)
p_type
p_offset
p_vaddr
p_paddr
p_filesz
p_memsz
p_flags
p_align

nasm vs. gas

References

[1] Essential C, Nick Parlante
[2] Efficient C Programming, Mark A. Weiss
[3] C A Reference Manual, Samuel P. Harbison & Guy L. Steele Jr.
[4] C Language Express, I. K. Chun
[5] “A Whirlwind Tutorial on Creating Really Teensy ELF Executables for Linux“
http://cseweb.ucsd.edu/~ricko/CSE131/teensyELF.htm
[6] http://en.wikipedia.org
[7] http://www.muppetlabs.com/~breadbox/software/tiny/teensy.html
[8] http://csapp.cs.cmu.edu/public/ch7-preview.pdf