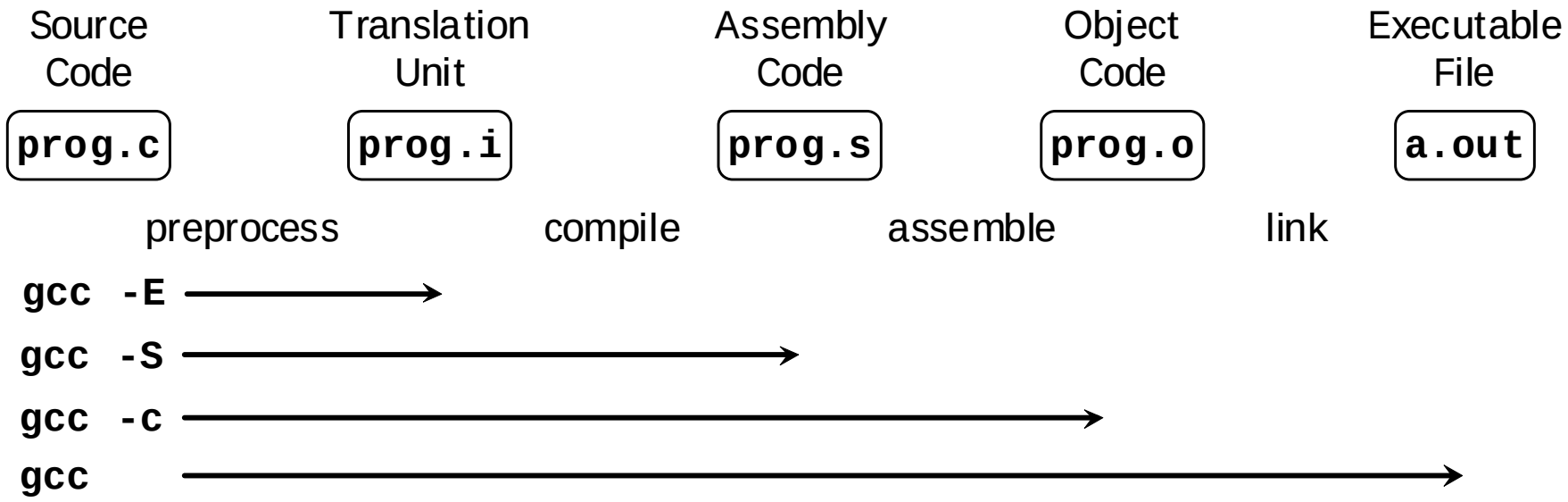
Linking Assembly with C

Stages of `gcc` operation

Stages of Gnu C compilation



~/asm$ gcc -S ex1 produces AT&T gas format assembly code

File **ex1.s**

Example — ex15

```
main()
{
        int i = 1 , j = 2 ;
        i = 2 * i;
        j = 4 * j;
}
```

Output of `gcc -S ex15`

```
.file   "ex15.c"
        .text
.globl main
        .type   main, @function
 pushl   %ebp                           ; push ebp
        movl    %esp, %ebp              ; mov ebp,esp
        subl    $16, %esp               ; sub esp,byte +0x10
        movl    $1, -4(%ebp)            ; mov dword [ebp-0x4],0x1
        movl    $2, -8(%ebp)            ; mov dword [ebp-0x8],0x2
        sall    -4(%ebp)                ; shl dword [ebp-0x4],1
        sall    $2, -8(%ebp)            ; shl dword [ebp-0x8],0x2
        leave
        ret
 .size   main, .-main
        .ident  "GCC: (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3"
        .section        .note.GNU-stack,"",@progbits
```

Utility program intel2gas

`intel2gas` can convert

gas to NASM (Intel)

```
intel2gas -g ex15.s
```

NASM to gas

NASM to gas inline assembler for C programs

Output of `intel2gas -g ex15.s`

```
 ;FILE "ex15.c"
        SECTION .text
GLOBAL main
        GLOBAL main:function
main:
        push    ebp
        mov     ebp,esp
        sub     esp,16
        mov     dword [ebp-4],1
        mov     dword [ebp-8],2
        sal     dword [ebp-4],1
        sal     dword [ebp-8],2
        leave
        ret
        GLOBAL  main:function (.-main)
        ;IDENT "GCC: (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3"
WARNING, Line 17: no syntax match: "    section   .note.GNU-stack,"",@progbits"
MISMATCH: "        .section        .note.GNU-stack,"",@progbits"
```

Warnings and mismatches

No NASM equivalent

Usually not critical

Linking Assembly to C — 1

factorial.c

```
#include <math.h>
#include <stdio.h>
main()
{
      int times;
      int i , j = 12;
      for (times = 0 ; times < 10000000 ; times++){
         i = factorial(j);
      }
      printf("%d\n",i);
}
int factorial(n)
      int n;
{
      if (n == 0)
         return 1;
      else
         return  n * factorial(n-1);
}
```

main
Calls **factorial(12)** 10,000,000 times
Long run time
Allows comparison with enhancements

factorial(n) calculates n! by recursion

Linking Assembly to C — 2

```
~/gcc$ gcc factorial.c -o factorial
```

Produces executable **`factorial`**

```
~/gcc$ time factorial
479001600

real    0m2.641s
user    0m2.524s
sys     0m0.004s
```

Program **`factorial`** runs in 2.528 seconds (user+system)

Ubuntu Linux server on 800 MHz Pentium III

Linking Assembly to C — 3

Compile program as separate files

factorial_a.c

```
main()
{
  int times;
  int i,j=12;
  for (times = 0 ; times < 10000000 ; times++){
    i = factorial(j);
  }
  printf("%d\n",i);
}
```

factorial_b.c

```
#include <math.h>
#include <stdio.h>

int factorial(n)
   int n;
{
   if (n == 0)
         return 1;
   else
     return  n * factorial(n-1);
}
```

Linking Assembly to C — 4

```
~/gcc$ gcc -c factorial_a.c
```

produces linkable object file **factorial_a.o**

```
~/gcc$ gcc -c factorial_b.c
```

produces linkable object file **factorial_b.o**

```
~/gcc$ gcc factorial_a.o factorial_b.o –o factorial
```

produces executable **factorial**

Identical to previous version

Linking Assembly to C — 5

Assembly output from `gcc -S factorial_a.c`

```
 .file   "factorial_a.c"
        .section        .rodata
.LC0:
        .string "%d\n"
        .text
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        andl    $-16, %esp
        subl    $32, %esp
        movl    $12, 20(%esp)
        movl    $0, 28(%esp)
        jmp     .L2
```

```
.L3:
        movl    20(%esp), %eax
        movl    %eax, (%esp)
        call    factorial
        movl    %eax, 24(%esp)
        addl    $1, 28(%esp)
.L2:
        cmpl    $9999999, 28(%esp)
        jle     .L3
        movl    $.LC0, %eax
        movl    24(%esp), %edx
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    printf
        leave
        ret
        .size   main, .-main
```

Linking Assembly to C — 6

Assembly output from `intel2gas -g factorial_a.s`

```
  ;FILE "factorial_a.c"
        SECTION         .rodata
.LC0:
        db      '%d',10,''
        SECTION .text
GLOBAL main
        GLOBAL main:function
main:
 push    ebp
 mov     ebp,esp
 and     esp,-16
 sub     esp,32
 mov     dword [esp+20],12 ; times
 mov     dword [esp+28],0  ; j
 jmp L2
```

```
L3:
 mov     eax, [esp+20] ; EAX <– times
 mov     [esp],eax     ; stack <– EAX
 call    factorial
 mov     [esp+24],eax     ; i <– return
 add     dword [esp+28],1 ; times++
L2:
        ; leave if times >= 10^7
 cmp     dword [esp+28],9999999
 jle L3
 mov     eax, .LC0
                ; EAX <–.LC0 = newline
 mov     edx, [esp+24]
                ; EDX <– i
 mov     [esp+4],edx
 mov     [esp],eax
        ; copy i, newline to stack
 call    printf
 leave
 ret
```

Linking Assembly to C — 7

Assembly version of `factorial` function written for `NASM`

Uses "register variables" to save memory accesses

Exploits advantages of Intel **`imul`** and **`loop`** instructions

```
; file factorial_c.asm
section .text
global factorial
factorial:
     push ebp              ; standard C data frame
     mov ebp,esp
     mov ecx,[ebp+8]       ; ECX <— parameter passes on stack
     mov eax,1             ; EAX <— 1
L1:  imul ecx              ; EAX <— EAX * ECX
     loop L1               ; ECX <— ECX - 1
                           ; if ECX <> 0 loop
     leave
     ret
```

Linking Assembly to C — 8

```
~/gcc$ nasm -f elf factorial_c.asm
```

produces linkable object file **factorial_c.o**

```
~/gcc$ gcc factorial_a.o factorial_c.o -o factorial_2
```

produces executable **factorial_2**

Identical to previous version

```
~/gcc$ time factorial_2
479001600

real    0m1.935s
user    0m1.828s
sys     0m0.008s
```

Program **factorial_2** runs in 1.836 seconds (user+system)

Faster by factor of (2.528 / 1.836) = 1.38

Using C Functions in Assembly Program

C functions

Defined in standard C libraries

Linked to program by **`gcc`**

Defined arguments and returns

Parameters passed on stack — right to left

```
return = function(arg1, arg2, arg3) ;
MOV [ESP], arg3
MOV [ESP+4], arg2
MOV [ESP+8], arg1
```

Return or pointer to list in **`EAX`**

Linking C functions to assembly

Assemble with NASM

Link using **`gcc`** with no options

Links C and Linux libraries to executable file

C function `printf()`

Formatted printing

```
printf("Hello World\n")
```

Prints

String **"Hello World" and newline character**

```
printf("%d%t%d%t", a, b)
printf("%f\n", c)
```

Prints

decimal_a<TAB>decimalb<TAB>float_c newline

```
printf("a = %d%t%b = d\n", a, b)
```

Prints

a = decimal_a<TAB>b = decimal_b newline

Using `printf()` in Assembly Program

```
        extern printf
        section .data
a:      dd     5
fmt:    db "a=%d, eax=%d", 10, 0   ; printf format string
                    ; printf("a=%d, eax=%d\n", a, a+2)
        section .text
        global main             ; C function – points EIP at main
main:   mov eax, [a]            ; EAX <– value of a
        add eax, 2              ; EAX <– EAX + 2
        push eax                ; value of a + 2
        push dword [a]          ; value of a
        push dword fmt          ; pointer to format string
        call printf             ; call C library function
        add esp, 12             ; clean up stack
                                ; (3 pushes of 4 bytes)
        call printf             ; call C library function
        mov eax,0               ; exit code
        ret                     ; C function – exits to Linux
```

Assemble + Link + Run

Assemble

```
~/asm$ nasm –f elf printf1.asm
```

Link

```
~/asm$ gcc printf1.o -o printf1
```

Run

```
~/asm$ printf1
a=5, eax=7
```

Exit code

```
~/asm $ echo $?
0
~/asm $
```

Another **printf** Example

```
        extern printf
        section .data
msg:    db      "Hello world: %c %s of length %d %d %X",10,0
char1:  db      'a'                 ; character a
str1:   db      "string",0          ; ASCIIZ string
len:    equ     $-str1              ; len = length of str1
inta1:  dd      1234567             ; integer 1234567
hex1:   dd      0x6789ABCD          ; hex constant
        section .text
        global main
main:   push dword [hex1]           ; %X - hex constant
        push dword [inta1]          ; %d - integer data
        push dword len              ; %d – constant (equate)
        push dword str1             ; %s – pointer to "string"
        push dword [char1]          ; %c – the character 'a'
        push dword msg              ; pointer to format string
        call printf                 ; call C library function
        add esp, 24                 ; pop stack 6*4 = 24 bytes
        mov eax, 0                  ; exit code
        ret
```

Assembly and Linking

Assemble

```
~/asm$ nasm -f elf printf2.asm
```

Link

```
~/asm$ gcc printf2.o -o printf2
```

Run

```
~/asm$ printf2
Hello world: a string of length 7 1234567 6789ABCD
~/asm$
```

Embedded Systems

Closed CPU-based devices

Not general purpose computers

99% of all microprocessors in embedded systems

Examples

Mouse, keyboard, Bluetooth earphone, microwave oven, blood pressure monitor, portable EKG, pedometer, toys, TV, DVD player, electronic doorbell, thermostat, remote control, car engine, ...

Economics

Cost = development budget + unit cost × number of units sold

Strategy = Spend more on development to get low unit cost

Minimum unit cost ⇒ cheapest, smallest CPU with least memory

Typical CPU in wireless mouse costs $0.25 per unit

Internal program memory ≈ 4 KB

Program in assembly and C

Small program to run fast on cheap CPU with small memory

Very Small C Program

c1.c

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

Compile

```
~/asm$ gcc c1.c -o c1
```

Run

```
~/asm$ c1 ; echo $?
42
```

Size of file

```
~/asm$ wc -c c1
7100 c1
~/asm$
```

Why So Big?

Program file contains 25 sections (from `objdump -s`)

Section	Bytes	Section	Bytes
Contents of section .interp:	19	Contents of section .text:	348
Contents of section .note.ABI-tag:	32	Contents of section .fini:	28
Contents of section .note.gnu.build-id:	36	Contents of section .rodata:	8
Contents of section .hash:	36	Contents of section .eh_frame:	4
Contents of section .gnu.hash:	32	Contents of section .ctors:	8
Contents of section .dynsym:	64	Contents of section .dtors:	8
Contents of section .dynstr:	69	Contents of section .jcr:	4
Contents of section .gnu.version:	8	Contents of section .dynamic:	208
Contents of section .gnu.version_r:	32	Contents of section .got:	4
Contents of section .rel.dyn:	8	Contents of section .got.plt:	20
Contents of section .rel.plt:	16	Contents of section .data:	8
Contents of section .init:	48	Contents of section .comment:	37
Contents of section .plt:	48	Header + strings + tail	5967

Same Program in Assembly

c2.asm

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

Assemble

```
~/asm$ nasm -f elf c2.asm
```

Link

```
~/asm$ gcc c2.o -o c2
```

Run

```
~/asm$ c2 ; echo $?
42
```

Size of file

```
~/asm$ wc -c c2
7102 c2
~/asm$
```

Volume = standard Linux + C library routines

Removing C Startup Files

c3.asm

```
EXTERN _exit          ; standard library function
GLOBAL _start         ; default EIP at start
                      ; does not link to C library
SECTION .text
_start:
        push dword 42
        call _exit
```

Assemble

```
~/asm$ nasm -f elf c3.asm
```

Link without C start-up routines

```
~/asm$ gcc -nostartfiles c3.o -o c3
```

Run

```
~/asm$ c3 ; echo $?
42
```

Size of file

```
~/asm$ wc -c c3
5363 c3
```

Removing Linux Library Files

c4.asm

```
GLOBAL _start          ; default EIP at start
                       ; does not link to C library
SECTION .text
_start:
        mov eax,1              ; Linux exit code
        mov ebx,42             ; exit code
        int 0x80               ; call Linux
```

Assemble

```
~/asm$ nasm -f elf c4.asm
```

Link without Linux Library Files

```
~/asm$ gcc -nostartfiles -nostdlib c4.o -o c4
```

Run

```
~/asm$ c4 ; echo $?
42
```

Size of file

```
~/asm$ wc -c c4
631 c4
```

Header + Program Sections

```
Header
0000000: 7f45 4c46 0101 0100 0000 0000 0000 0000  .ELF............
0000010: 0200 0300 0100 0000 a080 0408 3400 0000  ............4...
0000020: e000 0000 0000 0000 3400 2000 0200 2800  ........4. ...(.
0000030: 0600 0300 0100 0000 0000 0000 0080 0408  ................
0000040: 0080 0408 ac00 0000 ac00 0000 0500 0000  ................
0000050: 0010 0000 0400 0000 7400 0000 7480 0408  ........t...t...
0000060: 7480 0408 2400 0000 2400 0000 0400 0000  t...$...$.......
0000070: 0400 0000

Contents of section .note.gnu.build-id:
0000070:           0400 0000 1400 0000 0300 0000  ................
0000080: 474e 5500 89cc 5ea4 e9ba 8621 e005 7697  GNU...^....!..v.
0000090: 7b75 d6c3 1b86 f2de 0000 0000 0000 0000  {u..............

Contents of section .text:
00000a0: b801 0000 00bb 2a00 0000 cd80            ......*.....
```

Program Tail

```
00000a0:                                 002e 7379  ......*.......sy
00000b0: 6d74 6162 002e 7374 7274 6162 002e 7368  mtab..strtab..sh
00000c0: 7374 7274 6162 002e 6e6f 7465 2e67 6e75  strtab..note.gnu
00000d0: 2e62 7569 6c64 2d69 6400 2e74 6578 7400  .build-id..text.
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100: 0000 0000 0000 0000 1b00 0000 0700 0000  ................
0000110: 0200 0000 7480 0408 7400 0000 2400 0000  ....t...t...$...
0000120: 0000 0000 0000 0000 0400 0000 0000 0000  ................
0000130: 2e00 0000 0100 0000 0600 0000 a080 0408  ................
0000140: a000 0000 0c00 0000 0000 0000 0000 0000  ................
0000150: 1000 0000 0000 0000 1100 0000 0300 0000  ................
0000160: 0000 0000 0000 0000 ac00 0000 3400 0000  ............4...
0000170: 0000 0000 0000 0000 0100 0000 0000 0000  ................
0000180: 0100 0000 0200 0000 0000 0000 0000 0000  ................
0000190: d001 0000 8000 0000 0500 0000 0400 0000  ................
00001a0: 0400 0000 1000 0000 0900 0000 0300 0000  ................
00001b0: 0000 0000 0000 0000 5002 0000 2700 0000  ........P...'...
00001c0: 0000 0000 0000 0000 0100 0000 0000 0000  ................
00001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001e0: 0000 0000 7480 0408 0000 0000 0300 0100  ....t...........
00001f0: 0000 0000 a080 0408 0000 0000 0300 0200  ................
0000200: 0100 0000 0000 0000 0000 0000 0400 f1ff  ................
0000210: 0800 0000 a080 0408 0000 0000 1000 0200  ................
0000220: 0f00 0000 ac90 0408 0000 0000 1000 f1ff  ................
0000230: 1b00 0000 ac90 0408 0000 0000 1000 f1ff  ................
0000240: 2200 0000 ac90 0408 0000 0000 1000 f1ff  "...............
0000250: 0063 342e 6173 6d00 5f73 7461 7274 005f  .c4.asm._start._
0000260: 5f62 7373 5f73 7461 7274 005f 6564 6174  _bss_start._edat
```