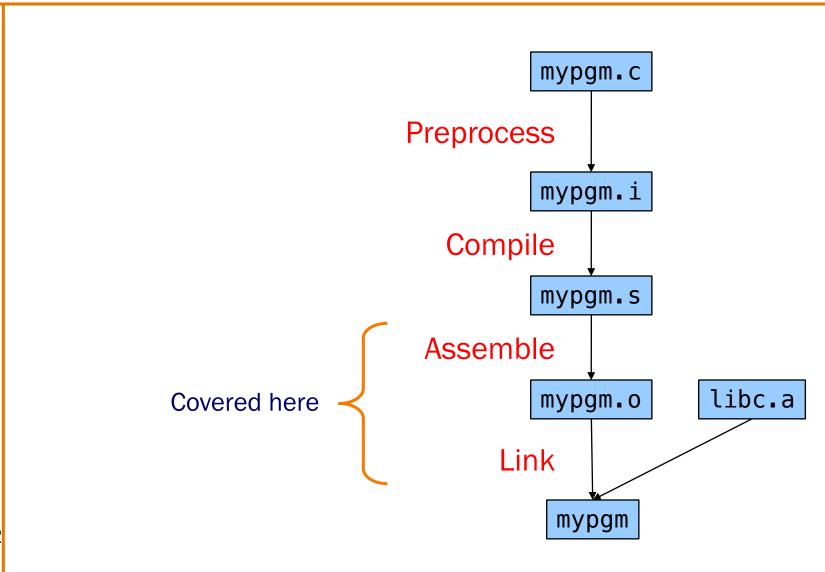
# COS 217: Introduction to Programming Systems

From Assembler to Linker



# The Build Process





2

# An Example Program



A simple (nonsensical) program, in C and assembly:

```
#include <stdio.h>
int main(void)
{  printf("Type a char: ");
  if (getchar() == 'A')
    printf("Hi\n");
  return 0;
}
```

Let's consider the machine language equivalent...

```
.section .rodata
msg1: .string "Type a char: "
msg2: .string "Hi\n"
      .section .text
      .global main
main:
      sub
               sp, sp, 16
               x30, [sp]
      str
      adr
               x0, msg1
      bl
               printf
      bl
               getchar
               w0, 'A'
      cmp
               skip
      bne
      adr
               x0, msg2
      bl
               printf
skip:
               w0, 0
      mov
               x30, [sp]
      ldr
      add
               sp, sp, 16
      ret
```

#### **Examining Machine Lang: RODATA**



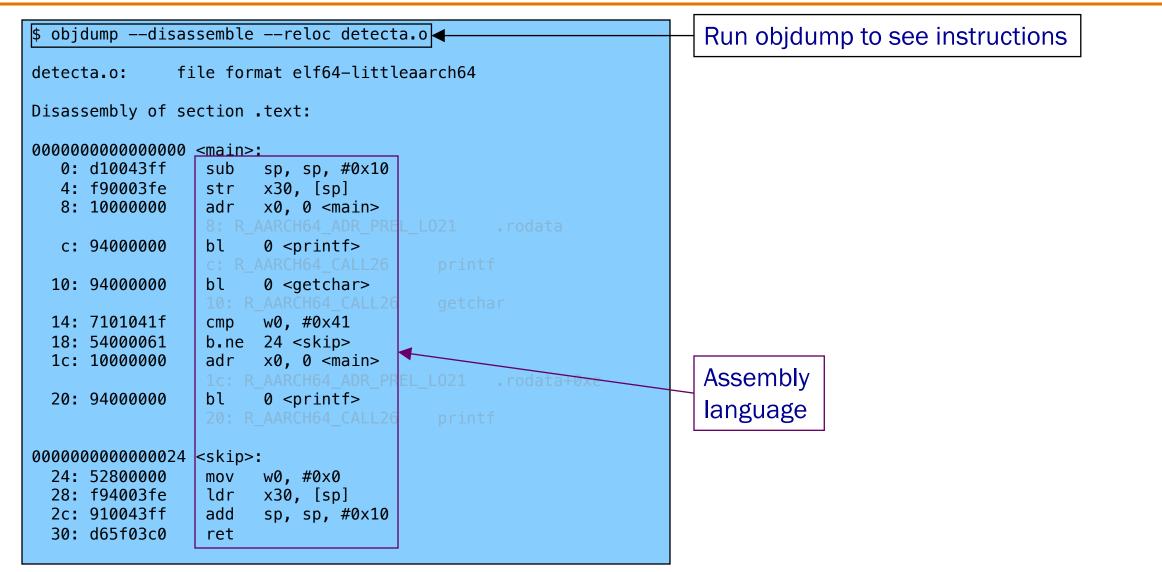
```
Assemble program; run objdump
$ gcc217 -c detecta.s
$ objdump --full-contents --section .rodata detecta.o
detecta.o: file format elf64-littleaarch64
Contents of section .rodata:
0000 54797065 20612063 6861723a 20004869 Type a char: .Hi
|0010 |0a00
  Offsets
             Contents

    Assembler does not know addresses

    Assembler knows only offsets

                               • "Type a char: " starts at offset 0x0
                                • "Hi\n" starts at offset Oxe
```







```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
0000000000000000 <main>:
   0: d10043ff
                  sub
                       sp, sp, #0x10
   4: f90003fe
                  str
                       x30, [sp]
                       x0, 0 <main>
   8: 10000000
                  adr
   c: 94000000
                        0 <printf>
                        0 <getchar>
  10: 94000000
                  cmp w_0, #0x41
  14: 7101041f
  18: 54000061 b.ne 24 <skip>
  1c: 100000000
                       x0, 0 <main>
  20: 94000000
                        0 <printf>
0000000000000024 <skip>:
  24: |52800000|
                       w0, #0x0
                  mov
  28: f94003fe
                  ldr
                       x30, [sp]
  2c: 910043ff
                  add
                        sp, sp, #0x10
  30: d65f03c0
                  ret
```

Run objdump to see instructions

Machine language



```
$ objdump --disassemble --reloc detecta.o ←
               file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
00000000000000000 <main>:
   0: d10043ff
                        sp, sp, #0x10
                  sub
   4: f90003fe
                  str
                        x30, [sp]
   8: 10000000
                        x0, 0 <main>
                         0 <printf>
   c: 94000000
                        % <getchar>
  10: 94000000
  14: 7101041<del>f</del>
                        w0, #0x41
                   cmp
  18: 54000061
                   b.ne 24 <skip>
  1c: 10000000
                         x0, 0 < main >
  20: 94000000
                         0 <printf>
0000000000000024 <skip>:
  24: 52800000
                        w0, #0x0
  28: f94003fe
                        x30, [sp]
  2c: 910043ff
                  add
                         sp, sp, #0x10
  30: d65f03c0
                   ret
```

Run objdump to see instructions

Offsets

Let's examine one line at a time...

sub sp, sp, #0x10



```
msb: bit 31
                                                                                    Isb: bit 0
$ objdump --d
                                                         sp, sp, #0x10
                                    0: d10043ff
                                                   sub
                1101 0001 0000 0000 0100 0011 1111 1111
detecta.o:
Disassembly o
000000000000000000000 <main>:
   0: d10043ff
                       sp, sp, #0x10
                 sub
   4: f90003fe
                       x30, [sp]
                 str
                       x0, 0 < main>
   8: 10000000
                 adr
   c: 94000000
                       0 <printf>
 10: 94000000
                       0 <getchar>
 14: 7101041f
                       w0, #0x41
                 cmp
 18: 54000061
                      24 <skip>
                 b.ne
 1c: 10000000
                       x0, 0 < main >
                       0 <printf>
 20: 94000000
0000000000000024 <skip>:
 24: 52800000
                       w0, #0x0
                 mov
 28: f94003fe
                 ldr
                       x30, [sp]
 2c: 910043ff
                 add
                       sp, sp, #0x10
 30: d65f03c0
                 ret
```

# sub sp, sp, #0x10



- opcode: subtract immediate
- Instruction width in bit 31: 1 = 64-bit
- Whether to set condition flags in bit 29: no
- Immediate value in bits 10-21:  $10000_b = 0x10 = 16$
- First source register in bits 5-9: 31 = sp
- Destination register in bits 0-4: 31 = sp
- Additional information about instruction: none

# str x30, [sp]



```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                  sub sp, sp, \#0x10
   4: f90003fe
                  str x30, [sp]
   8: 10000000
                       x0, 0 <main>
                  adr
   c: 94000000
                        0 <printf>
  10: 94000000
                        0 <getchar>
  14: 7101041f
                       w0, #0x41
                  cmp
  18: 54000061
                  b.ne 24 <skip>
  1c: 10000000
                       x0, 0 <main>
  20: 94000000
                        0 <printf>
0000000000000024 <skip>:
  24: 52800000
                       w0, #0x0
                 mov
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                  add
                        sp, sp, #0x10
  30: d65f03c0
                  ret
```

# str x30, [sp]



```
msb: bit 31 4: f90003fe str x30, [sp] lsb: bit 0 1111 1001 0000 0000 0000 0011 1111 1110
```

- opcode: store, register + offset
- Instruction width in bits 30-31: 11 = 64-bit
- Offset value in bits 12-20: 0
- "Source" (really destination) register in bits 5-9: 31 = sp
- "Destination" (really source) register in bits 0-4: 30
- Additional information about instruction: none

#### adr x0, 0 < main >



```
$ objdump --disassemble --reloc detecta.o
             file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
00000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, \#0x10
  4: f90003fe
                 str x30, [sp]
                       x0, 0 <main>
   8: 10000000
                 adr
   c: 94000000
                       0 <printf>
  10: 94000000
                       0 <getchar>
  14: 7101041f
                       w0, #0x41
                 cmp
  18: 54000061
                 b.ne 24 <skip>
  1c: 10000000
                       x0, 0 <main>
                       0 <printf>
  20: 94000000
0000000000000024 <skip>:
  24: 52800000
                 mov w0, #0x0
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                 ret
```

#### adr x0, 0 < main >



- opcode: generate address
- 19 High-order bits of relative address in bits 5-23: 0
- 2 Low-order bits of relative address in bits 29-30: 0
- Relative data location is 0 bytes after this instruction
- Destination register in bits 0-4:0
- Huh? That's not where msg1 lives!
  - Assembler knew that msg1 is a label within the RODATA section
  - But assembler didn't know address of RODATA section!
  - So, assembler couldn't generate this instruction completely, left a placeholder, and will request help from the linker



```
$ objdump --disassemble --reloc detecta.o
                                                                     Run objdump to see instructions
               file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                        sp, sp, #0x10
                  sub
   4: f90003fe
                  str x30, [sp]
   8: 10000000
                        x0.0 < main>
                  8: R AARCH64 ADR PREL L021
                                                . rodata
                        0 <printf>
   c: 94000000
                  c: R_AARCH64_CALL26
                                                                     Relocation
  10: 94000000
                        0 <getchar>
                  10: R AARCH64 CALL26
                                          getchar <
                                                                     records
  14: 7101041f
                        w0, #0x41
                  b.ne 24 < skip >
  18: 54000061
  1c: 10000000
                        x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_L021
                                                .rodata+0xe
  20: 94000000
                        0 <printf>
                                          printf
                  20: R AARCH64 CALL26
0000000000000024 <skip>:
  24: 52800000
                       w0, #0x0
                  mov
  28: f94003fe
                  ldr
                      x30, [sp]
  2c: 910043ff
                  add
                        sp, sp, #0x10
  30: d65f03c0
                  ret
```

# R\_AARCH64\_ADR\_PREL\_L021 .rodata



```
$ objdump --disassemble --reloc detecta.o
             file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
00000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, #0x10
  4: f90003fe str x30, [sp]
  8: 10000000
                 adr x0, 0 < main >
                 8: R AARCH64 ADR PREL L021 .rodata
                       0 <printf>
   c: 94000000
                                         printf
                 c: R AARCH64 CALL26
                       0 <getchar>
  10: 94000000
                                         getchar
                 10: R AARCH64 CALL26
  14: 7101041f
                       w0, #0x41
  18: 54000061
                 b.ne 24 < skip >
  1c: 10000000
                       x0, 0 < main >
                 1c: R_AARCH64_ADR_PREL_L021 .rodata+0xe
  20: 94000000
                       0 <printf>
                 20: R AARCH64 CALL26
                                         printf
0000000000000024 <skip>:
  24: 52800000
                 mov w0, \#0\times0
  28: f94003fe
                ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                 ret
```

#### Relocation Record 1



8: R\_AARCH64\_ADR\_PREL\_L021 rodata

This part is always the same, it's the name of the machine architecture!

#### Dear Linker,

Please patch the TEXT section at offset <u>0x8</u>. Patch in a <u>21</u>-bit\* signed offset of an address, <u>rel</u>ative to the <u>PC</u>, as appropriate for the <u>adr</u> instruction format. When you determine the address of <u>.rodata</u>, use that to compute the offset you need to do the patch.

Sincerely, Assembler

\* 19 High-order bits of relative address in bits 5-23
 2 Low-order bits of relative address in bits 29-30

# bl 0 <printf>



```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, \#0x10
  4: f90003fe
                 str x30, [sp]
   8: 10000000
                 adr x0, 0 < main >
                 8: R AARCH64 ADR PREL L021
                                               .rodata
                       0 <printf>
   c: 94000000
                                         printf
                  c: R_AARCH64_CALL26
  10: 94000000
                       0 <getchar>
                  10: R AARCH64 CALL26
                                         getchar
  14: 7101041f
                       w0, #0x41
                  b.ne 24 < skip >
  18: 54000061
  1c: 10000000
                       x0, 0 <main>
                 1c: R_AARCH64_ADR_PREL_L021
                                               .rodata+0xe
  20: 94000000
                       0 <printf>
                  bl
                  20: R AARCH64 CALL26
                                         printf
0000000000000024 <skip>:
  24: 52800000
                       w0, #0x0
                 mov
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                  ret
```

# bl 0 <printf>



- opcode: branch and link
- Relative address in bits 0-25: 0
- Huh? That's not where printf lives!
  - Assembler had to calculate [addr of printf] [addr of this instr]
  - But assembler didn't know address of printf it's off in some library (libc) and isn't present (yet)!
  - So, assembler couldn't generate this instruction completely, left a placeholder, and will request help from the linker

### R\_AARCH64\_CALL26

# printf



```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                  sub sp, sp, \#0x10
   4: f90003fe
                 str x30, [sp]
   8: 10000000
                  adr x0, 0 < main >
                  8: R AARCH64 ADR PREL L021
                                                .rodata
                        0 <printf>
   c: 94000000
                                          printf
                  c: R AARCH64 CALL26
                        0 <getchar>
  10: 94000000
                  10: R AARCH64 CALL26
                                          getchar
  14: 7101041f
                       w0, #0x41
  18: 54000061
                  b.ne 24 < skip >
  1c: 10000000
                       x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_L021
                                               .rodata+0xe
  20: 94000000
                        0 <printf>
                  20: R AARCH64 CALL26
                                          printf
0000000000000024 <skip>:
  24: 52800000
                       w0, #0x0
                 mov
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                  add
                        sp, sp, #0x10
  30: d65f03c0
                  ret
```

#### Relocation Record 2



c: R\_AARCH64\_CALL26 printf

Dear Linker,

Please patch the TEXT section at offset Oxc. Patch in a 26-bit signed offset relative to the PC, appropriate for the function call (bl) instruction format. When you determine the address of printf, use that to compute the offset you need to do the patch.

Sincerely, Assembler

# bl 0 <getchar>



```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, \#0x10
  4: f90003fe
                 str x30, [sp]
   8: 10000000
                       x0, 0 < main >
                 adr
                 8: R AARCH64 ADR PREL L021
                                               .rodata
                       0 <printf>
   c: 94000000
                                          printf
                  c: R_AARCH64_CALL26
                       0 <getchar>
  10: 94000000
                  10: R AARCH64 CALL26
                                         getchar
  14: 7101041f
                       w0, #0x41
  18: 54000061
                  b.ne 24 < skip >
  1c: 10000000
                       x0, 0 <main>
                 1c: R_AARCH64_ADR_PREL_L021
                                               .rodata+0xe
  20: 94000000
                       0 <printf>
                  20: R AARCH64 CALL26
                                          printf
0000000000000024 <skip>:
  24: 52800000
                       w0, #0x0
                 mov
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                  ret
```

# bl 0 <getchar>



- opcode: branch and link
- Relative address in bits 0-25: 0
- Same situation as before relocation record coming up!

#### Relocation Record 3



10: R\_AARCH64\_CALL26 getchar

Dear Linker,

Please patch the TEXT section at offset 0x10. Patch in a 26-bit signed offset relative to the PC, appropriate for the function  $\underline{call}$  (bl) instruction format. When you determine the address of  $\underline{getchar}$ , use that to compute the offset you need to do the patch.

Sincerely, Assembler

### cmp w0, $\#0\times41$



```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, \#0x10
  4: f90003fe
                 str x30, [sp]
   8: 10000000
                 adr x0, 0 < main >
                 8: R AARCH64 ADR PREL L021
                                              . rodata
                       0 <printf>
   c: 94000000
                                         printf
                 c: R_AARCH64_CALL26
                       0 <getchar>
  10: 94000000
                 10: R AARCH64 CALL26
                                         getchar
  14: 7101041f
                       w0, #0x41
                 cmp
                 b.ne 24 <skip>
  18: 54000061
  1c: 10000000
                       x0, 0 <main>
                 1c: R_AARCH64_ADR_PREL_L021 .rodata+0xe
  20: 94000000
                       0 <printf>
                 bl
                 20: R AARCH64 CALL26
                                         printf
0000000000000024 <skip>:
  24: 52800000
                 mov w0, #0x0
  28: f94003fe
                ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                 ret
```

#### cmp w0, $\#0\times41$



- Recall that cmp is really an assembler alias:
   this is the same instruction as subs wzr, w0, 0x41
- opcode: subtract immediate
- Instruction width in bit 31: 0 = 32-bit
- Whether to set condition flags in bit 29: yes
- Immediate value in bits 10-21:  $1000001_b = 0x41 = 'A'$
- First source register in bits 5-9: 0
- Destination register in bits 0-4: 31 = wzr
  - Note that register #31 (11111 $_{\rm b}$ ) is used to mean either sp or xzr/wzr, depending on the instruction

# b.ne 24 <skip>



```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
000000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, \#0x10
  4: f90003fe
                 str x30, [sp]
   8: 10000000
                 adr x0, 0 < main >
                 8: R AARCH64 ADR PREL L021
                                               .rodata
                       0 <printf>
   c: 94000000
                                         printf
                 c: R_AARCH64_CALL26
                       0 <getchar>
  10: 94000000
                 10: R AARCH64 CALL26
                                         getchar
  14: 7101041f
                       w0, #0x41
                 cmp
                 b.ne 24 < skip >
  18: 54000061
  1c: 10000000
                       x0, 0 <main>
                 1c: R_AARCH64_ADR_PREL_L021 .rodata+0xe
  20: 94000000
                       0 <printf>
                 bl
                 20: R AARCH64 CALL26
                                         printf
0000000000000024 <skip>:
  24: 52800000
                 mov w0, #0x0
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                 ret
```

# b.ne 24 <skip>



- This instruction is at offset 0x18, and skip is at offset 0x24, which is 0x24 0x18 = 0xc = 12 bytes later
- opcode: conditional branch
- Relative address in bits 5-23:  $11_b$ . Shift left by 2:  $1100_b = 12$
- Conditional branch type in bits 0-4: NE
- No need for relocation record!
  - Assembler had to calculate [addr of skip] [addr of this instr]
  - Assembler did know offsets of skip and this instruction
  - So, assembler could generate this instruction completely, and does not need to request help from the linker





```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
00000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, #0x10
  4: f90003fe
                 str x30, [sp]
  8: 10000000
                 adr x0, 0 < main >
                 8: R AARCH64 ADR PREL L021
                                           .rodata
                       0 <printf>
   c: 94000000
                 c: R AARCH64 CALL26
                                        printf
  10: 94000000
                       0 <getchar>
                 10: R AARCH64 CALL26
                                        getchar
  14: 7101041f
                       w0, #0x41
  18: 54000061
                 b.ne 24 < skip >
  1c: 10000000
                 adr x0, 0 < main >
                 1c: R AARCH64 ADR PREL L021 .rodata+0xe
  20: 94000000
                       0 <printf>
                 20: R AARCH64 CALL26
                                        printf
0000000000000024 <skip>:
  24: 52800000
                 mov w0, \#0\times0
  28: f94003fe
                ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                 ret
```

#### Relocation Record 4



Dear Linker,

Please patch the TEXT section at offset Ox1c.

Patch in a 21-bit signed offset of an address, relative to the PC, as appropriate for the adr instruction format. When you determine the address of rodata, add Oxe and use that to compute the offset you need to do the patch.

Sincerely, Assembler





```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
00000000000000000000 <main>:
   0: d10043ff
                 sub sp, sp, #0x10
  4: f90003fe
                 str x30, [sp]
   8: 10000000
                 adr x0, 0 < main >
                 8: R AARCH64 ADR PREL L021
                                               .rodata
                       0 <printf>
   c: 94000000
                 c: R AARCH64 CALL26
                                         printf
  10: 94000000
                       0 <getchar>
                 10: R AARCH64 CALL26
                                         getchar
  14: 7101041f
                       w0, #0x41
  18: 54000061
                 b.ne 24 < skip >
  1c: 10000000
                       x0, 0 <main>
                 1c: R_AARCH64_ADR_PREL_L021
                                               .rodata+0xe
                       0 <printf>
  20: 94000000
                 20: R AARCH64 CALL26
                                         printf
0000000000000024 <skip>:
  24: 52800000
                       w0, #0x0
                 mov
  28: f94003fe
                 ldr x30, [sp]
  2c: 910043ff
                 add
                       sp, sp, #0x10
  30: d65f03c0
                 ret
```

# Last Example: Your Turn!



What does this relocation record mean?

20: 94000000 bl 0 <printf>

20: R\_AARCH64\_CALL26 printf

See context on previous slides with parallel records:

bl printf (#48)

bl getchar (#51)

Dear Linker,

Please patch the TEXT section at offset <u>0x20</u>. Patch in a <u>26</u>-bit signed offset relative to the PC, appropriate for the function <u>call</u> (bl) instruction format. When you determine the address of <u>printf</u>, use that to compute the offset you need to do the patch.

Sincerely, Assembler





```
$ objdump --disassemble --reloc detecta.o
              file format elf64-littleaarch64
detecta.o:
Disassembly of section .text:
00000000000000000 <main>:
   0: d10043ff
                 sub
                        sp, sp, #0x10
   4: f90003fe
                       x30, [sp]
                  str
   8: 10000000
                        x0, 0 < main >
                  adr
                  8: R AARCH64 ADR PREL L021
                                                .rodata
                        0 <printf>
   c: 94000000
                  c: R AARCH64 CALL26
                                          printf
 10: 94000000
                        0 <qetchar>
                  10: R AARCH64 CALL26
                                          getchar
 14: 7101041f
                        w0, #0x41
 18: 54000061
                  b.ne 24 < skip >
 1c: 10000000
                        x0, 0 < main >
                  1c: R_AARCH64_ADR_PREL_L021
                                                . rodata+0xe
 20: 94000000
                        0 <printf>
                  20: R AARCH64 CALL26
                                          printf
0000000000000024 <skip>:
 24: 52800000
                       w0, #0x0
                  mov
 28: f94003fe
                 ldr x30, [sp]
 2c: 910043ff
                  add
                        sp, sp, #0x10
 30: d65f03c0
                  ret
```

#### Exercise for you:

using information from these slides, create a bitwise breakdown of these instructions, and convince yourself that the hex values are correct!

#### From Assembler to Linker



Assembler writes its data structures to .o file

#### Linker:

- Reads .o file
- Writes executable binary file
- Works in two phases: resolution and relocation

#### Linker Resolution



#### Resolution

Linker resolves references

#### For our sample program, linker:

- Notes that labels getchar and printf are unresolved
- Fetches machine language code defining getchar and printf from libc.a
- Adds that code to TEXT section
- Adds more code (e.g. definition of \_start) to TEXT section too
- Adds code to other sections too

#### **Linker Relocation**





#### Relocation

- Linker relocates code into final combined sections with addresses
- Linker traverses relocation records, patching instructions as specified

@impatrickt

### Examining Machine Language: RODATA



Link program; run objdump on final exectuable

```
$ gcc217 detecta.o -o detecta

$ objdump --full-contents --section .rodata detecta

detecta: file format elf64-littleaarch64

Contents of section .rodata:

400710 01000200 00000000 000000000 00000000

400720 54797065 20612063 6861723a 20004869 Type a char: .Hi

400730 0a00 ...
```

Addresses, not offsets

RODATA is at 0x400710
Starts with some header info
Real start of RODATA is at 0x400720
"Type a char: " starts at 0x400720
"Hi\n" starts at 0x40072e

## **Examining Machine Language: TEXT**



```
$ objdump --disassemble --reloc detecta
             file format elf64-littleaarch64
detecta:
      . . .
0000000000400650 <main>:
  400650:
            d10043ff
                              sp, sp, #0x10
                        sub
  400654:
            f90003fe
                             x30, [sp]
                        str
                        adr x0, 400720 <msq1>
            10000640
  400658:
           97ffffa1
                              4004e0 <printf@plt>
  40065c:
           97ffff9c
                        bl
                              4004d0 <getchar@plt>
  400660:
           7101041f
  400664:
                        cmp
                              w0, #0x41
            54000061
                        b.ne 400674 <skip>
  400668:
            50000600
                              x0, 40072e < msq2 >
  40066c:
                        adr
  400670:
            97ffff9c
                        bl
                              4004e0 <printf@plt>
0000000000400674 <skip>:
  400674:
            52800000
                              w0, #0x0
                        mov
            f94003fe
                             x30, [sp]
  400678:
                        ldr
           910043ff
                              sp, sp, #0x10
  40067c:
                        add
            d65f03c0
  400680:
                        ret
                       Addresses,
                       not offsets
```

Run objdump to see instructions





```
$ objdump --disassemble --reloc detecta
             file format elf64-littleaarch64
detecta:
      . . .
                                                                               Additional code
0000000000400650 <main>:
  400650:
           d10043ff
                             sp, sp, #0x10
                        sub
  400654:
           f90003fe
                        str
                             x30, [sp]
                        adr x0, 400720 <msq1>
  400658:
           10000640
  40065c:
          97ffffa1
                             4004e0 <printf@plt>
  400660:
          97ffff9c
                        bl
                             4004d0 <getchar@plt>
           7101041f
                             w0, #0x41
  400664:
                        cmp
                        b.ne 400674 <skip>
  400668:
           54000061
                             x0, 40072e <msg2>
           50000600
  40066c:
                        adr
                              4004e0 <printf@ptt>
  400670:
           97ffff9c
                        bl
0000000000400674 <skip>:
  400674:
            52800000
                             w0, #0x0
                        mov
  400678:
           f94003fe
                        ldr
                              x30/[sp]
  40067c:
           910043ff
                        add
                              sp, sp, #0x10
  400680:
           d65f03c0
                        ret
```

## **Examining Machine Language: TEXT**



```
$ objdump --disassemble --reloc detecta
             file format elf64-littleaarch64
detecta:
      . . .
0000000000400650 <main>:
  400650:
           d10043ff
                              sp, sp, #0x10
                        sub
  400654:
           f90003fe
                             x30, [sp]
                        str
                        adr x0.400720 < msq1 >
  400658:
           10000640
  40065c:
          97ffffa1
                             4004e0 <printf@plt>
  400660:
          97ffff9c
                             4004d0 <getchar@plt>
           7101041f
  400664:
                        cmp
                             w0, #0x41
           54000061
                        b.ne 400674 <skip>
  400668:
          50000600
                             x0, 40072e < msg2 >
  40066c:
                        adr
  400670:
           97ffff9c
                        bl
                              4004e0 <printf@plt>
0000000000400674 <skip>:
  400674:
           52800000
                              w0, #0x0
                        mov
  400678:
          f94003fe
                             x30, [sp]
                        ldr
  40067c:
          910043ff
                              sp, sp, #0x10
                        add
  400680:
           d65f03c0
                        ret
```



No relocation records!

Let's see what the linker did with them...

40

# adr x0, 400720 < msg1 >



```
$ objdump --disassemble --reloc detecta
            file format elf64-littleaarch64
detecta:
      . . .
0000000000400650 <main>:
  400650:
           d10043ff
                             sp, sp, #0x10
                       sub
                             x30, [sp]
  400654:
           f90003fe
                       str
  400658:
           10000640
                             x0, 400720 <msq1>
                       adr
  40065c:
          97ffffa1
                       bl
                             4004e0 <printf@plt>
  400660:
           97ffff9c
                       bl
                             4004d0 <getchar@plt>
           7101041f
                             w0, #0x41
  400664:
                       cmp
           54000061
                            400674 <skip>
  400668:
                       b.ne
  40066c:
           50000600
                             x0, 40072e <msq2>
                       adr
  400670:
           97ffff9c
                       bl
                             4004e0 <printf@plt>
0000000000400674 <skip>:
  400674:
           52800000
                             w0, #0x0
                       mov
  400678:
           f94003fe
                             x30, [sp]
                       ldr
  40067c:
          910043ff
                             sp, sp, #0x10
                       add
  400680:
           d65f03c0
                        ret
```

## adr x0, 400720 < msg1 >



- opcode: generate address
- 19 High-order bits of offset in bits 5-23: 110010
- 2 Low-order bits of offset in bits 29-30: 00
- Relative data location is 11001000b = 0xc8 bytes after this instruction
- Destination register in bits 0-4:0
- msg1 is at 0x400720
- this instruction is at 0x400658
- 0x400720 0x400658 = 0xc8

### bl

# 4004e0 <printf@plt>



```
$ objdump --disassemble --reloc detecta
             file format elf64-littleaarch64
detecta:
      . . .
0000000000400650 <main>:
  400650:
            d10043ff
                              sp, sp, #0x10
                        sub
                              x30, [sp]
  400654:
           f90003fe
                        str
  400658:
           10000640
                              \times 0, 400720 < msq1 >
                        adr
  40065c:
          97ffffa1
                              4004e0 <printf@plt>
  400660:
           97ffff9c
                        bl
                              4004d0 <getchar@plt>
           7101041f
                              w0, #0x41
  400664:
                        cmp
                             400674 <skip>
  400668:
           54000061
                        b.ne
            50000600
  40066c:
                              x0, 40072e < msg2 >
                        adr
  400670:
            97ffff9c
                        bl
                              4004e0 <printf@plt>
0000000000400674 <skip>:
  400674:
            52800000
                              w0, #0x0
                        mov
  400678:
           f94003fe
                              x30, [sp]
                        ldr
  40067c:
          910043ff
                              sp, sp, #0x10
                        add
  400680:
           d65f03c0
                        ret
```

## bl 4

# 4004e0 <pri>deplt>



```
msb: bit 31 40065c: 97ffffa1 bl 4004e0 <pri>rintf@plt> | lsb: bit 0 1001 0111 1111 1111 1111 1111 1010 0001
```

- opcode: branch and link
- Relative address in bits 0-25: 26-bit two's complement of  $1011111_b$ . But remember to shift left by two bits (see earlier slides)! This gives  $-101111100_b = -0x17c$
- printf is at 0x4004e0
- this instruction is at 0x40065c
- 0x4004e0 0x40065c = -0x17c





```
$ objdump --disassemble --reloc detecta
            file format elf64-littleaarch64
detecta:
      . . .
0000000000400650 <main>:
  400650:
           d10043ff
                             sp, sp, #0x10
                       sub
                             x30, [sp]
  400654:
           f90003fe
                       str
  400658:
           10000640
                             x0, 400720 <msq1>
                       adr
  40065c:
          97ffffa1
                       bl
                             4004e0 <printf@plt>
  400660:
          97ffff9c
                       bl
                             4004d0 <getchar@plt>
           7101041f
                             w0, #0x41
  400664:
                       cmp
           54000061
                       b.ne 400674 <skip>
  400668:
  40066c:
           50000600
                             x0, 40072e <msq2>
                       adr
  400670:
           97ffff9c
                       bl
                             4004e0 <printf@plt>
0000000000400674 <skip>:
  400674:
           52800000
                             w0, #0x0
                       mov
  400678:
          f94003fe
                             x30, [sp]
                       ldr
  40067c:
          910043ff
                             sp, sp, #0x10
                       add
  400680:
           d65f03c0
                       ret
```

# Summary



#### AARCH64 Machine Language

- 32-bit instructions
- Formats have conventional locations for opcodes, registers, etc.

#### **Assembler**

- Reads assembly language file
- Generates TEXT, RODATA, DATA, BSS sections
  - Containing machine language code
- Generates relocation records
- Writes object (.o) file

#### Linker

- Reads object (.o) file(s)
- Does resolution: resolves references to make code complete
- Does relocation: traverses relocation records to patch code
- Writes executable binary file

# Wrapping Up the Course



No precepts today/tomorrow – this is the last class meeting.

Assignment 6 is due on Wednesday 5/1

"Regular" assignment – not subject to Dean's Date restrictions

Plenty of office hours over the next 2 weeks

Exact schedule has been announced and pinned on Ed

Review session: Wednesday 5/8 at 2:00 PM in CS 104

Final Exam: Friday 5/10 at 9:00 AM in McCosh 10

https://www.cs.princeton.edu/courses/cos217/exam2.php

### We Have Covered:



### Programming in the large

- Program design
- Programming style
- Building
- Testing
- Debugging
- Data structures
- Modularity
- Performance
- Version control

### Programming at several levels

- ARM Machine Language
- ARM Assembly Language
- The C programming language
- (just a taste of) the bash shell

### Core systems and organization ideas

- Preprocess, Compile, Assemble, Link
- Storage hierarchy
- (just a taste of) Processes and VM



return EXIT\_SUCCESS;

The end.