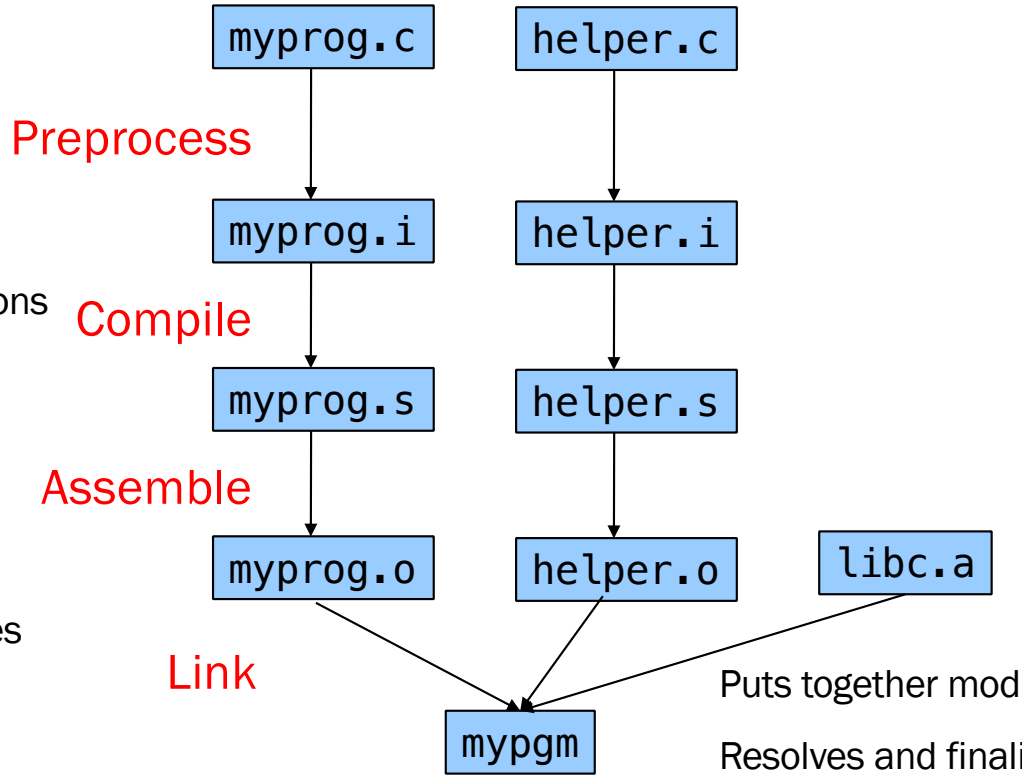
COS 217: Introduction to Programming Systems

From Assembler to Linker

PRINCETON UNIVERSITY

The Build Process



Translates assembly instructions to machine instructions

Puts in temporary labels for locations in memory segments, not actual addresses, since it knows only its local addresses and segments

Doesn't know where they will finally land when put together

Puts in labels to external addresses it doesn't know (library functions)

Leaves "love notes" for linker to please fill them in correctly

Puts together modules and libraries

Resolves and finalizes addresses

Produces final executable

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;
}
```

Assembler knows relative locations of the labels in this file, but not final location of RODATA section, and not location of printf code from libarary

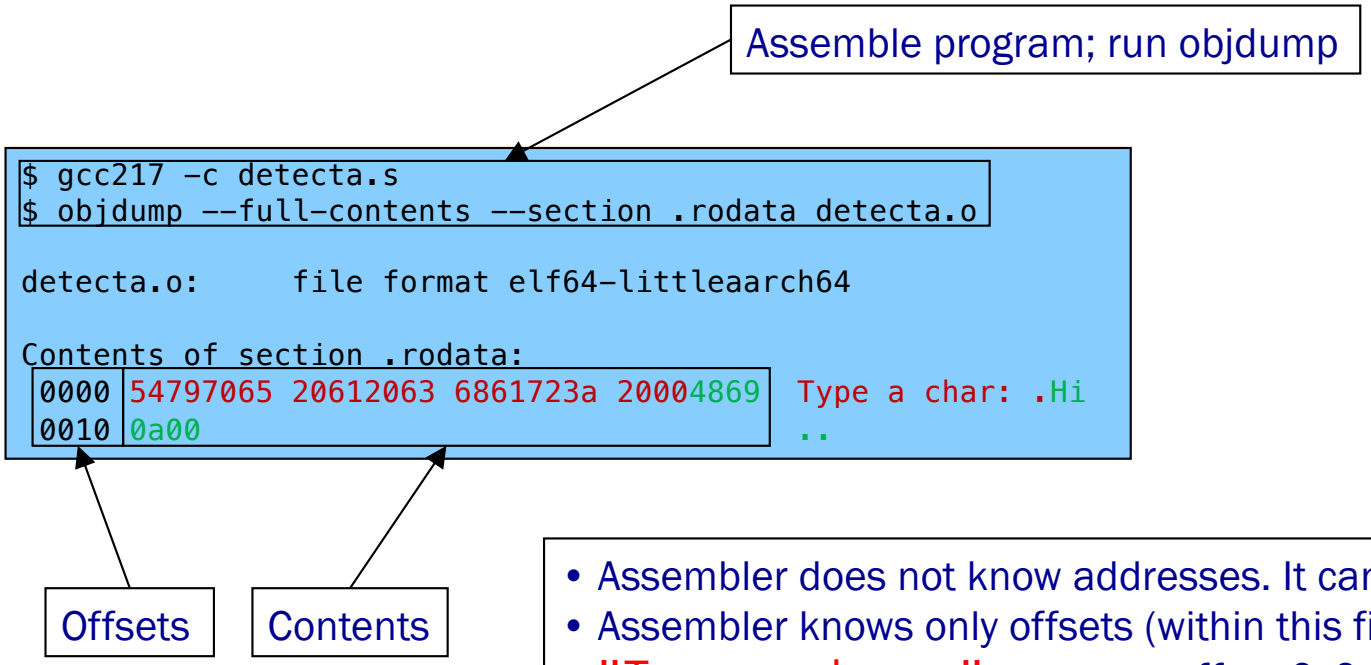
```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

References label in its RODATA section

References label in its text section

References external label

Examining Machine Language: RODATA



- Assembler does not know addresses. It cannot set them.
- Assembler knows only offsets (within this file's RODATA)
- `"Type a char: "` starts at offset 0x0. Obviously not final addr.
- `"Hi\n"` starts at offset 0xe, since "Type a char: " is 14 or 0xe chars
- Actual addresses of the RODATA sections of files will be determined by linker

Examining Machine Language: TEXT

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0x0
  20: 94000000    bl    0 <printf>
                  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
```

Run objdump to see instructions

Assembly language

Examining Machine Language: TEXT

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26    printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21    .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Run objdump to see instructions

Machine language

Examining Machine Language: TEXT

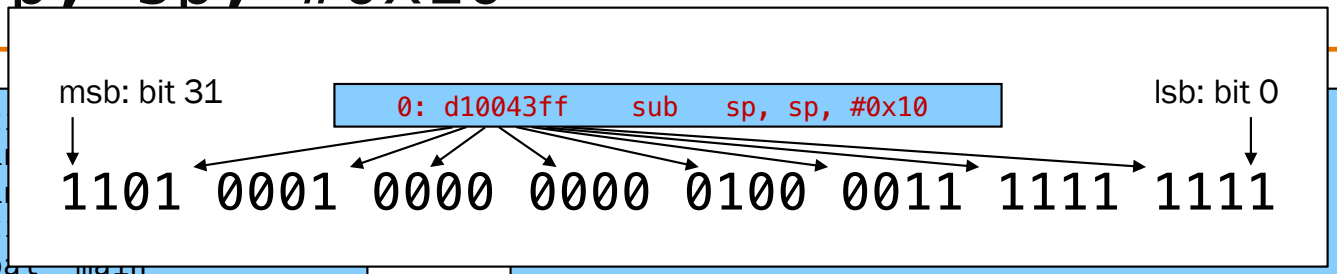
```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26    printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21    .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Run objdump to see instructions

Offsets, from beginning of this file's text section: Assembler can't determine actual addresses, e.g. due to conflicts in addresses when assembling different files.

Let's examine one line at a time…

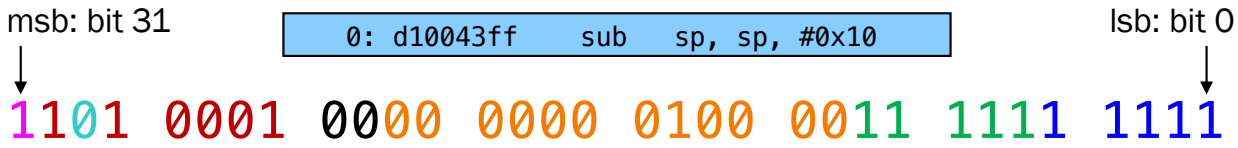
sub sp, sp, #0x10




```
        .sect
msg1:   .stri
msg2:   .stri
        .sect
        .globa  main
main:
        sub     sp, sp, 16
        str     x30, [sp]

        adr     x0, msg1
        bl      printf

        bl      getchar
        cmp     w0, 'A'
        bne     skip

        adr     x0, msg2
        bl      printf
skip:
        mov     w0, 0
        ldr     x30, [sp]
        add     sp, sp, 16
        ret
```

```
0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

sub sp, sp, #0x10



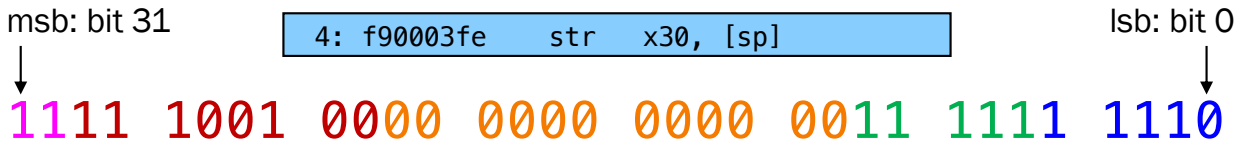
- opcode: subtract immediate
- Instruction width in bit 31: 1 = 64-bit
- Whether to set condition flags in bit 29: no
- Immediate value in bits 10-21: 10000_b = 0x10 = 16
- First source register in bits 5-9: 31 = sp
- Destination register in bits 0-4: 31 = sp
- Additional information about instruction: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

str x30, [sp]

```
        .section .rodata
msg1:   .string  "Type a char: "
msg2:   .string  "Hi\n"
        .section .text
        .global  main
main:
        sub      sp, sp, 16
        str      x30, [sp]

        adr      x0, msg1
        bl       printf

        bl       getchar
        cmp      w0, 'A'
        bne      skip

        adr      x0, msg2
        bl       printf
skip:
        mov      w0, 0
        ldr      x30, [sp]
        add      sp, sp, 16
        ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

str x30, [sp]



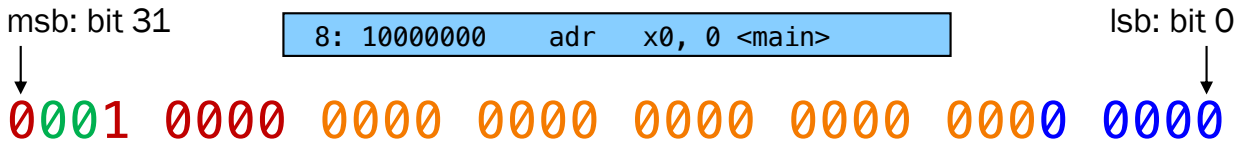
- opcode: store, register + offset
- Instruction width in bits 30-31: 11 = 64-bit
- Offset value in bits 12-20: 0
- “Source” (really destination) register in bits 5-9: 31 = sp
- “Destination” (really source) register in bits 0-4: 30
- Additional information about instruction: none

adr x0, 0 <main>

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

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 this instruction. And that's not where `msg1` lives in memory
  - Assembler knew `msg1` is a label in RODATA section, but didn't know start address of RODATA
  - So, assembler couldn't generate this instruction completely.
  - So it left a placeholder, and will request help from the linker

Examining Machine Language: TEXT

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Relocation records

R_AARCH64_ADR_PREL_LO21 .rodata

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Relocation Record 1

```
8: R_AARCH64_ADR_PREL_LO21    .rodata
```

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

Dear Linker,

Please patch the TEXT section at offset 0x8.
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, use that to compute the offset you need to do the patch.

Sincerely,
Assembler

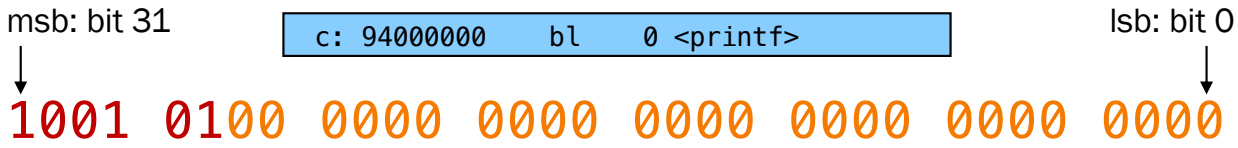
* 19 High-order bits of relative address in bits 5-23
2 Low-order bits of relative address in bits 29-30

bl 0 <printf>

```
        .section .rodata
msg1:   .string  "Type a char: "
msg2:   .string  "Hi\n"
        .section .text
        .global  main
main:
        sub      sp, sp, 16
        str      x30, [sp]

        adr      x0, msg1
        bl       printf

        bl       getchar
        cmp      w0, 'A'
        bne      skip

        adr      x0, msg2
        bl       printf
skip:
        mov      w0, 0
        ldr      x30, [sp]
        add      sp, sp, 16
        ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

`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

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Relocation Record 2

```
c: R_AARCH64_CALL26     printf
```

Dear Linker,

Please patch the TEXT section at offset 0xc. 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.

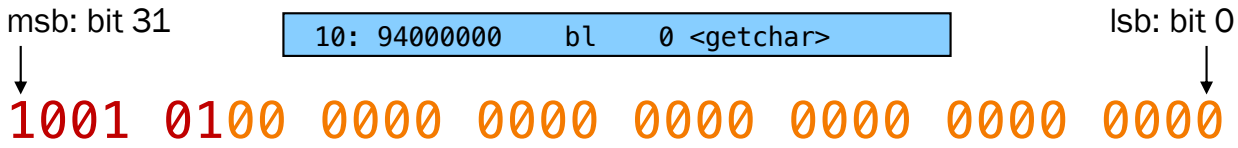
Sincerely,
Assembler

bl 0 <getchar>

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

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 call (bl) instruction format.
When you determine the address of getchar, use that
to compute the offset you need to do the patch.

Sincerely,
Assembler

cmp w0, #0x41

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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, #0x41`

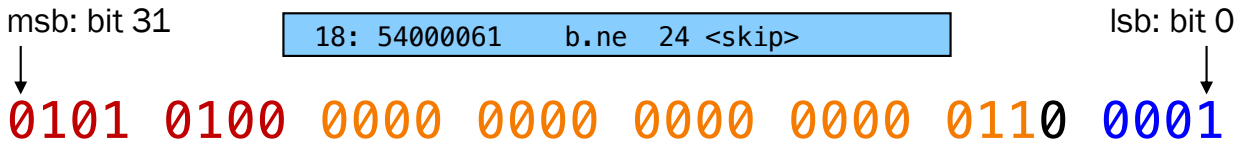
- Recall that `cmp` is really an assembler alias:
  this is the same instruction as `subs wzr, w0, 0x41`
- opcode: subtract immediate
- Instruction width in bit 31: 0 = 32-bit
- Whether to set condition flags in bit 29: yes
- Immediate value in bits 10-21: 1000001_b = 0x41 = 'A'
- First source register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

b.ne 24 <skip>

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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

R_AARCH64_ADR_PREL_LO21 .rodata+0xe

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Relocation Record 4

```
1c: R_AARCH64_ADR_PREL_LO21    .rodata+0xe
```

Dear Linker,

Please patch the TEXT section at offset 0x1c.
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 0xe and use that to compute the offset you need to do the patch.

Sincerely,
Assembler

Another printf, with relocation record...

```
      .section .rodata
msg1: .string  "Type a char: "
msg2: .string  "Hi\n"
      .section .text
      .global  main
main:
      sub      sp, sp, 16
      str      x30, [sp]

      adr      x0, msg1
      bl       printf

      bl       getchar
      cmp      w0, 'A'
      bne      skip

      adr      x0, msg2
      bl       printf
skip:
      mov      w0, 0
      ldr      x30, [sp]
      add      sp, sp, 16
      ret
```

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Everything Else is Similar...

```
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-littleaarch64

Disassembly of section .text:

0000000000000000 <main>:
   0: d10043ff    sub   sp, sp, #0x10
   4: f90003fe    str   x30, [sp]
   8: 10000000    adr   x0, 0 <main>
                  8: R_AARCH64_ADR_PREL_LO21    .rodata
   c: 94000000    bl    0 <printf>
                  c: R_AARCH64_CALL26     printf
  10: 94000000    bl    0 <getchar>
                  10: R_AARCH64_CALL26    getchar
  14: 7101041f    cmp   w0, #0x41
  18: 54000061    b.ne  24 <skip>
  1c: 10000000    adr   x0, 0 <main>
                  1c: R_AARCH64_ADR_PREL_LO21   .rodata+0xe
  20: 94000000    bl    0 <printf>
                  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
```

Exercise:
using information from the 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

Linker resolves references: every reference in object file is related to a single symbol definition in the input files (same file or a different file)

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

If it doesn't find the function ("`prontf()`", when compiling prontf.c):

```
$ gcc217 prontf.c
prontf.c: In function 'main':
prontf.c:6:1: warning: implicit declaration of function 'prontf' [-Wimplicit-function-declaration]
 {  prontf("hello, world\n");
 ^
/tmp/ccjA2CnG.o: In function `main':
prontf.c:(.text+0x10): undefined reference to `prontf'
collect2: error: ld returned 1 exit status
```

Linker Relocation

@impatrickt

Relocation

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

Finally, Let's Examine the 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 00000000 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 the Machine Language: TEXT

```
$ objdump --disassemble --reloc detecta

detecta:     file format elf64-littleaarch64

      ...

0000000000400650 <main>:
  400650:   d10043ff    sub   sp, sp, #0x10
  400654:   f90003fe    str   x30, [sp]
  400658:   10000640    adr   x0, 400720 <msg1>
  40065c:   97ffffa1    bl    4004e0 <printf@plt>
  400660:   97ffff9c    bl    4004d0 <getchar@plt>
  400664:   7101041f    cmp   w0, #0x41
  400668:   54000061    b.ne  400674 <skip>
  40066c:   50000600    adr   x0, 40072e <msg2>
  400670:   97ffff9c    bl    4004e0 <printf@plt>

0000000000400674 <skip>:
  400674:   52800000    mov   w0, #0x0
  400678:   f94003fe    ldr   x30, [sp]
  40067c:   910043ff    add   sp, sp, #0x10
  400680:   d65f03c0    ret
```

Run objdump to see instructions

Addresses, not offsets

IN the assembly code, main was at offset 0 and skip at 0x24 in text section

Here, still 0x24 apart: just the absolute addresses are different (full addresses)

Examining the Machine Language: TEXT

```
$ objdump --disassemble --reloc detecta

detecta:     file format elf64-littleaarch64

      ...

0000000000400650 <main>:
  400650:   d10043ff    sub   sp, sp, #0x10
  400654:   f90003fe    str   x30, [sp]
  400658:   10000640    adr   x0, 400720 <msg1>
  40065c:   97ffffa1    bl    4004e0 <printf@plt>
  400660:   97ffff9c    bl    4004d0 <getchar@plt>
  400664:   7101041f    cmp   w0, #0x41
  400668:   54000061    b.ne  400674 <skip>
  40066c:   50000600    adr   x0, 40072e <msg2>
  400670:   97ffff9c    bl    4004e0 <printf@plt>

0000000000400674 <skip>:
  400674:   52800000    mov   w0, #0x0
  400678:   f94003fe    ldr   x30, [sp]
  40067c:   910043ff    add   sp, sp, #0x10
  400680:   d65f03c0    ret
```

Additional code

Examining the Machine Language: TEXT

```
$ objdump --disassemble --reloc detecta

detecta:     file format elf64-littleaarch64

      ...

0000000000400650 <main>:
  400650:   d10043ff    sub   sp, sp, #0x10
  400654:   f90003fe    str   x30, [sp]
  400658:   10000640    adr   x0, 400720 <msg1>
  40065c:   97ffffa1    bl    4004e0 <printf@plt>
  400660:   97ffff9c    bl    4004d0 <getchar@plt>
  400664:   7101041f    cmp   w0, #0x41
  400668:   54000061    b.ne  400674 <skip>
  40066c:   50000600    adr   x0, 40072e <msg2>
  400670:   97ffff9c    bl    4004e0 <printf@plt>

0000000000400674 <skip>:
  400674:   52800000    mov   w0, #0x0
  400678:   f94003fe    ldr   x30, [sp]
  40067c:   910043ff    add   sp, sp, #0x10
  400680:   d65f03c0    ret
```

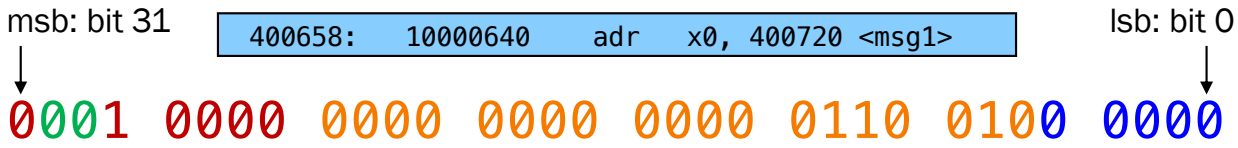
No relocation records!

Let's see what the linker did with them…

adr x0, 400720 <msg1>

```
$ objdump --disassemble --reloc detecta

detecta:     file format elf64-littleaarch64

      ...

0000000000400650 <main>:
  400650:   d10043ff    sub   sp, sp, #0x10
  400654:   f90003fe    str   x30, [sp]
  400658:   10000640    adr   x0, 400720 <msg1>
  40065c:   97ffffa1    bl    4004e0 <printf@plt>
  400660:   97ffff9c    bl    4004d0 <getchar@plt>
  400664:   7101041f    cmp   w0, #0x41
  400668:   54000061    b.ne  400674 <skip>
  40066c:   50000600    adr   x0, 40072e <msg2>
  400670:   97ffff9c    bl    4004e0 <printf@plt>

0000000000400674 <skip>:
  400674:   52800000    mov   w0, #0x0
  400678:   f94003fe    ldr   x30, [sp]
  40067c:   910043ff    add   sp, sp, #0x10
  400680:   d65f03c0    ret
```

adr x0, 400720 <msg1>



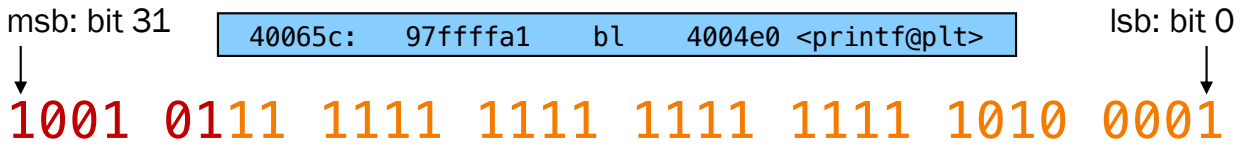
- opcode: generate address
- 19 High-order bits of offset in bits 5-23: 110010
- 2 Low-order bits of offset in bits 29-30: 00
- *Relative* data location is 11001000b = 0xc8 bytes after this instruction
- Destination register in bits 0-4:0

- msg1 is at 0x400720
- this instruction is at 0x400658
- 0x400720 – 0x400658 = 0xc8 ✓

bl 4004e0 <printf@plt>

```
$ objdump --disassemble --reloc detecta

detecta:     file format elf64-littleaarch64

      ...

0000000000400650 <main>:
  400650:   d10043ff    sub   sp, sp, #0x10
  400654:   f90003fe    str   x30, [sp]
  400658:   10000640    adr   x0, 400720 <msg1>
  40065c:   97ffffa1    bl    4004e0 <printf@plt>
  400660:   97ffff9c    bl    4004d0 <getchar@plt>
  400664:   7101041f    cmp   w0, #0x41
  400668:   54000061    b.ne  400674 <skip>
  40066c:   50000600    adr   x0, 40072e <msg2>
  400670:   97ffff9c    bl    4004e0 <printf@plt>

0000000000400674 <skip>:
  400674:   52800000    mov   w0, #0x0
  400678:   f94003fe    ldr   x30, [sp]
  40067c:   910043ff    add   sp, sp, #0x10
  400680:   d65f03c0    ret
```

bl 4004e0 <printf@plt>



- 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 $-1\ 0111\ 1100_b = -0x17c$

- `printf` is at 0x4004e0
- this instruction is at 0x40065c
- 0x4004e0 – 0x40065c = –0x17c ✓

Everything Else is Similar…

```
$ objdump --disassemble --reloc detecta

detecta:     file format elf64-littleaarch64

      ...

0000000000400650 <main>:
  400650:   d10043ff    sub   sp, sp, #0x10
  400654:   f90003fe    str   x30, [sp]
  400658:   10000640    adr   x0, 400720 <msg1>
  40065c:   97ffffa1    bl    4004e0 <printf@plt>
  400660:   97ffff9c    bl    4004d0 <getchar@plt>
  400664:   7101041f    cmp   w0, #0x41
  400668:   54000061    b.ne  400674 <skip>
  40066c:   50000600    adr   x0, 40072e <msg2>
  400670:   97ffff9c    bl    4004e0 <printf@plt>

0000000000400674 <skip>:
  400674:   52800000    mov   w0, #0x0
  400678:   f94003fe    ldr   x30, [sp]
  40067c:   910043ff    add   sp, sp, #0x10
  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

We Have, in COS217, 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

COS217.c

```
main() {
    learn_some_core_concepts();
    program_in_C_and_assembly();
    return 0;
}
```