# CSCI 2321 (Computer Design), Spring 2018

## Assemble/Link Example

This is a completed version of the example we partly did in class 2/21, meant as an example of what I ask you to in the assemble/link problem in Homework 3.

It may be useful to keep in mind the goal: Some instructions can be completely translated to machine code at assembly time, but the ones involving "absolute addresses" cannot. The goal of this problem was to show you some of what a real assembler and linker would do to translate such instructions. Keep in mind that assemblers typically produce a block of machine code (the "text segment"), a block of data (the "data segment"), and some tables that will be useful in linking them together. Linkers work with this output to produce combined text and data segments, filling in the instructions that had to be left incomplete at assembly time.

For this example, I'm using the factorial program, split into two files and with a few tweaks to make the problem either easier or more interesting. The first file is the main function with all its data areas; the second file is a stripped-down version of the factorial function with one bogus variable to illustrate how merging data areas works.

**Assembly phase.** The first step is to do the relevant parts of assembling each file. I like to use a stepwise procedure:

- Copy the original file, remove comments, expand pseudoinstructions as needed, and write down offsets for each instruction and data item. For instructions that reference absolute addresses, I like to put in a "?" for the values I don't know, and then an indication of the symbol being referenced.
- From this you can get the lengths of the two segments and build a symbol table (all code/data labels defined in the file) and a relocation table with information about the instructions that will need to be filled in ("patched") at link time.

**File 1 (main function).** The first step gives you this:

```
text segment:

     main:
0000         addi    $sp, $sp, -4
0004         sw      $ra, 0($sp)
#       la      $a0, prompt
0008         lui $at, ? [ prompt ]
000c         ori $a0, $at, ? [ prompt ]
0010         li      $v0, 4          # "print string" syscall
0014         syscall
0018         li      $v0, 5          # "read int" syscall
001c         syscall
0020         addi    $s0, $v0, 0     # save result in $s0
0024         addi    $a0, $s0, 0     # parameter is input
0028         jal     ? [ factorial ]
002c         addi    $s1, $v0, 0     # copy return value
#         la      $a0, answer_txt1
0030         lui $at, ? [ answer_txt1 ]
```

```
0034         ori $a0, $at, ? [ answer_txt1 ]
0038         li      $v0, 4          # "print string" syscall
003c         syscall
0040         addi    $a0, $s0, 0     # get saved input
0044         li      $v0, 1          # "print int" syscall
0048         syscall
#         la      $a0, answer_txt2
004c         lui $at, ? [ answer_txt2 ]
0050         ori $a0, $at, ? [ answer_txt2 ]
0054         li      $v0, 4          # "print string" syscall
0058         syscall
005c         addi    $a0, $s1, 0     # get saved result
0060         li      $v0, 1          # "print int" syscall
0064         syscall
#         la      $a0, nl
0068         lui $at, ? [ nl ]
006c         ori $a0, $at, ? [ nl ]
0070         li      $v0, 4          # "print string" syscall
0074         syscall
0078         lw      $ra, 0($sp)
007c         addi    $sp, $sp, 4
0080         jr      $ra
             .end    main
(length 0x84)

data segment (notice that .asciiz adds a null character at the end)

0000 prompt: .asciiz "Enter an integer:\n" (19 chars)
0013 answer_txt1: .asciiz "factorial of " (14 chars)
0021 answer_txt2: .asciiz " is " (5 chars)
0026 nl:     .asciiz "\n" (2 chars)

(length 0x20)
```

which gives you the sizes of the text and data segments and what you need for the other two tables:

```
symbol table:

main            0000, text
prompt          0000, data
answer_txt1     0013, data
answer_txt2     0021, data
nl              0026, data

relocation table:

0008   lui $at, ?              prompt
000c   ori $a0, $at, ?         prompt
```

```
0028   jal ?                    factorial
0030   lui $at, ?               answer_txt1
0034   ori $a0, $at, ?          answer_txt1
004c   lui $at, ?               answer_txt2
0050   ori $a0, $at, ?          answer_txt2
0068   lui $at, ?               nl
006c   ori $a0, $at, ?          nl
```

**File 2 (factorial function).** The first step gives you this:

```
text segment:

     factorial:
0000         addi    $sp, $sp, -4
0004         sw      $ra, 0($sp)
# la    $t0, dummy
0008         lui $at, ? [ dummy ]
000c         ori $t0, $at, ? [ dummy ]
0010         li $v0, 1
0014         lw      $ra, 0($sp)
0018         addi    $sp, $sp, 4
001c         jr      $ra

(length 0x20)

data segment:

0000 dummy: .word 0

(length 0x4)
```

which gives you the sizes of the text and data segments and what you need for the other two tables:

```
symbol table:

factorial       0000, text
dummy           0000, data

relocation table:

0008   lui $at, ?               dummy
000c   ori $t0, $at, ?          dummy
```

**Link phase.** The next step is to do the relevant parts of linking the two files, with the main objective being to fill in those instructions that referenced absolute addresses. I like to start by just merging the code and data segments from the assembly phase, replacing offsets with addresses. That gives this:

```
text segment, starting at 0x0040 0024

     main:
0x0040 0024         addi    $sp, $sp, -4
0x0040 0028         sw      $ra, 0($sp)
#       la      $a0, prompt
0x0040 002c         lui $at, ? [ prompt ]
0x0040 0030         ori $a0, $at, ? [ prompt ]
0x0040 0034         li      $v0, 4          # "print string" syscall
0x0040 0038         syscall
0x0040 003c         li      $v0, 5          # "read int" syscall
0x0040 0040         syscall
0x0040 0044         addi    $s0, $v0, 0     # save result in $s0
0x0040 0048         addi    $a0, $s0, 0     # parameter is input
0x0040 004c         jal     ? [ factorial ]
0x0040 0050         addi    $s1, $v0, 0     # copy return value
#         la      $a0, answer_txt1
0x0040 0054         lui $at, ? [ answer_txt1 ]
0x0040 0058         ori $a0, $at, ? [ answer_txt1 ]
0x0040 005c         li      $v0, 4          # "print string" syscall
0x0040 0060         syscall
0x0040 0064         addi    $a0, $s0, 0     # get saved input
0x0040 0068         li      $v0, 1          # "print int" syscall
0x0040 006c         syscall
#         la      $a0, answer_txt2
0x0040 0070         lui $at, ? [ answer_txt2 ]
0x0040 0074         ori $a0, $at, ? [ answer_txt2 ]
0x0040 0078         li      $v0, 4          # "print string" syscall
0x0040 007c         syscall
0x0040 0080         addi    $a0, $s1, 0     # get saved result
0x0040 0084         li      $v0, 1          # "print int" syscall
0x0040 0088         syscall
#         la      $a0, nl
0x0040 008c         lui $at, ? [ nl ]
0x0040 0090         ori $a0, $at, ? [ nl ]
0x0040 0094         li      $v0, 4          # "print string" syscall
0x0040 0098         syscall
0x0040 009c         lw      $ra, 0($sp)
0x0040 00a0         addi    $sp, $sp, 4
0x0040 00a4         jr      $ra
             .end    main
     factorial:
0x0040 00a8         addi    $sp, $sp, -4
0x0040 00ac         sw      $ra, 0($sp)
# la    $t0, dummy
0x0040 00b0         lui $at, ? [ dummy ]
0x0040 00b4         ori $t0, $at, ? [ dummy ]
0x0040 00b8         li  $v0, 1
```

```
0x0040 00bc         lw      $ra, 0($sp)
0x0040 00c0         addi    $sp, $sp, 4
0x0040 00c4         jr      $ra

(length 0xc8, which is 0x84 + 0x24)

data segment, starting at 0x1001 0000

0x1001 0000 prompt: .asciiz "Enter an integer:\n" (19 chars)
0x1001 0013 answer_txt1: .asciiz "factorial of " (14 chars)
0x1001 0021 answer_txt2: .asciiz " is " (5 chars)
0x1001 0026 nl:     .asciiz "\n" (2 chars)
0x1001 0028 dummy:  .word 0

(length 0x2c, which is 0x28 + 0x04)
```

which gives you the sizes of the text and data segments and let you build a combined symbol table (this time showing absolute addresses only):

```
symbol table:

main            0x0040 0024
factorial       0x0040 00a8
prompt          0x1001 0000
answer_txt1     0x1001 0013
answer_txt2     0x1001 0021
nl              0x1001 0026
dummy           0x1001 0028
```

To actually patch the instructions, for `jal` we replace the "?" with the address from the symbol table, while for the `lui` and `ori` we replace the "?" with the upper and lower 16 bits, respectively, of the address from the symbol table, giving this:

```
patched instructions:

0x0040 002c         lui $at, 0x1001 [ prompt ]
0x0040 0030         ori $a0, $at, 0x0000 [ prompt ]
0x0040 004c         jal 0x0040 00a8 [ factorial ]
0x0040 0054         lui $at, 0x1001 [ answer_txt1 ]
0x0040 0058         ori $a0, $at, 0x0013 [ answer_txt1 ]
0x0040 0070         lui $at, 0x1001 [ answer_txt2 ]
0x0040 0074         ori $a0, $at, 0x0021 [ answer_txt2 ]
0x0040 008c         lui $at, 0x1001 [ nl ]
0x0040 0090         ori $a0, $at, 0x0026 [ nl ]
0x0040 00b0         lui $at, 0x1001 [ dummy ]
0x0040 00b4         ori $t0, $at, 0x0068 [ dummy ]
```

Whew!! This is a lot of details and arithmetic, and it's easy to get wrong, but if I check it with SPIM, first loading file 1 and then file 2, and stepping through . . . All the "patched" instructions seem to be right!!