Example RISC-V Assembly Programs

Source: Stephen Marz at University of Tennessee, Knoxville

Contents

Most of these instructions do not use *aliases* (aka pseudo instructions) so that you can test your ISA emulator without the assembler changing instructions on you. However, some instructions you will see use aliases, such as la (load address) since we need the assembler to fill in the memory address of a given symbol.
We will also discuss the choices of instructions and flow of program in class. Make sure you follow each element, and then we will discuss why I chose not to do a 1-for-1 translation of C-to-assembly in some of these programs.

When testing your emulator, test the following by assembling using the given linker script org.lds.

String Length

Determine the length of a C-style string by adding 1 until we find the terminator ‘\0’.

```
int strlen(const char *str) {
int i;
for (i = 0;str[i] != '\0';i++);
return i;
}

.section .text
.global strlen
strlen:
# a0 = const char *str
add t0, zero, zero # i = 0
1: # Start of for loop
```

```
add t1, t0, a0 # Add the byte offset for str[i]
lb t1, 0(t1) # Dereference str[i]
beq t1, zero, 1f # if str[i] == 0, break for loop
addi t0, t0, 1 # Add 1 to our iterator
jal zero, 1b # Jump back to condition (1 backwards)
1: # End of for loop
addi a0, t0, 0 # Move t0 into a0 to return
jalr zero, ra # Return back via the return address register
```

String Copy

Copy one C-style string into another. I expanded the actual implementation to make it easier to see each element in assembly.

```
void stringcopy(char *dst, const char *src) {
do {
// Copy the source's byte into the destination's byte
*dst = *src;
// We check '\0' here so that we copy the source's \0
// into the destination.
if (*src == '\0') break;
// Advance both the destination and source to the next byte.
dst++;
src++;
} while (true);
}
```

```
.section .text
.global stringcopy
stringcopy:
# a0 = destination
# a1 = source
1:
lb t0, 0(a1) # Load a char from the src
sb t0, 0(a0) # Store the value of the src
beq t0, zero, 1f # Check if it's 0
addi a0, a0, 1 # Advance destination one byte
```

```
addi a1, a1, 1 # Advance source one byte
jal zero, 1b # Go back to the start of the loop
1:
jalr zero, 0(ra) # Return back via the return address
```

String Copy (n bytes)

```
// from the strncpy man pages
char *strncpy(char *dst, const char *src, unsigned long n) {
unsigned long i;
for (i = 0; i < n && src[i] != '\0'; i++)
dst[i] = src[i];
for ( ; i < n; i++)
dst[i] = '\0';
return dst;
}
```

```
.section .text
.global strncpy
strncpy:
# a0 = char *dst
# a1 = const char *src
# a2 = unsigned long n
# t0 = i
addi t0, zero, 0 # i = 0
1: # first for loop
bge t0, a2, 1f # break if i >= n
add t1, a1, t0 # src + i
lb t1, 0(t1) # t1 = src[i]
beq t1, zero, 1f # break if src[i] == '\0'
add t2, a0, t0 # t2 = dst + i
sb t1, 0(t2) # dst[i] = src[i]
addi t0, t0, 1 # i++
jal zero, 1b # back to beginning of loop
1: # second for loop
bge t0, a2, 1f # break if i >= n
```

```
add t1, a0, t0 # t1 = dst + i
sb zero, 0(t1) # dst[i] = 0
addi t0, t0, 1 # i++
jal zero, 1b # back to beginning of loop
1:
# we don't have to move anything since
# a0 hasn't changed.
jalr zero, 0(ra) # return via return address register
```

Reverse a string

```
void strrev(char *str) {
int i;
int sz = strlen(str);
for (i = 0;i < sz / 2;i++) {
char c = str[i];
str[i] = str[sz - i - 1];
str[sz - i - 1] = c;
}
}
```

```
.section .text
.global strrev
strrev:
# s1 = str
# a0 = sz
# t0 = sz / 2
# t1 = i
# Enter stack frame
addi sp, sp, -16
sd ra, 0(sp)
sd s1, 8(sp)
# Get the size of the string
mv s1, a0
call strlen
srai t0, a0, 1 # Divide sz by 2
```

```
li t1, 0 # i = 0
1: # for loop
bge t1, t0, 1f
add t2, s1, t1 # str + i
sub t3, a0, t1 # sz - i
addi t3, t3, -1 # sz - i - 1
add t3, t3, s1 # str + sz - i - 1
lb t4, 0(t2) # str[i]
lb t5, 0(t3) # str[sz - i - 1]
sb t4, 0(t3) # swap
sb t5, 0(t2)
addi t1, t1, 1
j 1b
1:
# Leave stack frame
ld s1, 8(sp)
ld ra, 0(sp)
addi sp, sp, 16
ret
```

Sum of an Integer Array

```
int arraysum(int a[], int size) {
int ret = 0;
int i;
for (i = 0;i < size;i++) {
ret = ret + a[i];
}
return ret;
}

.section .text
.global arraysum
arraysum:
# a0 = int a[]
# a1 = int size
```

```
# t0 = ret
# t1 = i
addi t0, zero, 0 # ret = 0
addi t1, zero, 0 # i = 0
1: # For loop
bge t1, a1, 1f # if i >= size, break
slli t2, t1, 2 # Multiply i by 4 (1 << 2 = 4)
add t2, a0, t2 # Update memory address
lw t2, 0(t2) # Dereference address to get integer
add t0, t0, t2 # Add integer value to ret
addi t1, t1, 1 # Increment the iterator
jal zero, 1b # Jump back to start of loop (1 backwards)
1:
addi a0, t0, 0 # Move t0 (ret) into a0
jalr zero, 0(ra) # Return via return address register
```

Bubble Sort

```
void bubsort(long *list, int size) {
bool swapped;
do {
swapped = false;
for (int i = 1;i < size;i++) {
if (list[i-1] > list[i]) {
swapped = true;
long tmp = list[i-1];
list[i-1] = list[i];
list[i] = tmp;
}
}
} while (swapped);
}
```

```
.section .text
.global bubsort
bubsort:
```

```
# a0 = long *list
# a1 = size
# t0 = swapped
# t1 = i
1: # do loop
addi t0, zero, 0 # swapped = false
addi t1, zero, 1 # i = 1
2: # for loop
bge t1, a1, 2f # break if i >= size
slli t3, t1, 3 # scale i by 8 (for long)
add t3, a0, t3 # new scaled memory address
ld t4, -8(t3) # load list[i-1] into t4
ld t5, 0(t3) # load list[i] into t5
ble t4, t5, 3f # if list[i-1] < list[i], it's in position
# if we get here, we need to swap
addi t0, zero, 1 # swapped = true
sd t4, 0(t3) # list[i] = list[i-1]
sd t5, -8(t3) # list[i-1] = list[i]
3: # bottom of for loop body
addi t1, t1, 1 # i++
jal zero, 2b # loop again
2: # bottom of do loop body
bne t0, zero, 1b # loop if swapped = true
jalr zero, 0(ra) # return via return address register
```

Distance Formula w/ 64-bit Floats

```
struct Coordinate {
double x;
double y;
};
double distance(const Coordinate &from, const Coordinate &to) {
double x = to.x - from.x;
double y = to.y - from.y;
return sqrt(x*x + y*y);
```

```
}
.section .text
.global distance
distance:
# a0 = Coordinate &from
# a1 = Coordinate &to
# Coordinate structure
# Name Offset Size (bytes)
# x 0 8
# y 8 8
# Total size = 16 bytes
fld ft0, 0(a0) # ft0 = from.x
fld ft1, 8(a0) # ft1 = from.y
fld ft2, 0(a1) # ft2 = to.x
fld ft3, 8(a1) # ft3 = to.y
fsub.d ft0, ft2, ft0 # ft0 = to.x - from.x
fsub.d ft1, ft3, ft1 # ft1 = to.y - from.y
fmul.d ft0, ft0, ft0 # ft0 = ft0 * ft0
fmul.d ft1, ft1, ft1 # ft1 = ft1 * ft1
fadd.d ft0, ft0, ft1 # ft0 = ft0 + ft1
fsqrt.d fa0, ft0 # fa0 = sqrt(ft0)
# Return value goes in fa0
jalr zero, 0(ra) # Return
```

Min Max w/ 32-bit Floats

```
void minmax(float a, float b, float c, float &mn, float &mx) {
mn = mx = a;
if (mn > b) mn = b;
else if (mx < b) mx = b;
if (mn > c) mn = c;
else if (mx < c) mx = c;
}
.section .text
.global minmax
```

```
minmax:
# fa0 = float a
# fa1 = float b
# fa2 = float c
# ft0 = min
# ft1 = max
# a0 = float &mn
# a1 = float &mx
# Set mn = mx = a
fmv.s ft0, fa0
fmv.s ft1, fa0
fgt.s t0, ft0, fa1 # if (mn > b)
beq t0, zero, 1f # skip if false
# if we get here, then mn is > b
fmv.s ft0, fa1 # ft0 is mn, set it to b
jal zero, 2f # jump past the else if statement
1:
flt.s t0, ft1, fa1 # else if (mx < b)
beq t0, zero, 2f # skip if false
# if we get here then mx < b
fmv.s ft1, fa1 # ft1 is mx, set it to b
2:
fgt.s t0, ft0, fa2 # if (mn > c)
beq t0, zero, 1f # skip if false
# if we get here then mn > c
fmv.s ft0, fa2 # ft0 is mn, set it to c
jal zero, 2f # skip the else if statement
1:
flt.s t0, ft1, fa2 # else if (mx < c)
beq t0, zero, 2f # skip if false
# If we get here then mx < c
fmv.s ft1, fa2 # ft1 is mx, set it to c
2:
fsw ft0, 0(a0) # store minimum into &mn
fsw ft1, 0(a1) # store maximum into &mx
jalr zero, 0(ra) # return via return address register
```

Add Element to Singly Linked List

```
LL *addll(LL *list, LL *element) {
element->next = list;
return element;
}
```

```
.section .text
.global addll
addll:
# a0 = list
# a1 = element
# LL structure
# Name Offset Size (bytes)
# data 0 2
# next 8 8
sd a0, 8(a1) # element->next = list
addi a0, a1, 0 # set a0 to return element instead of list
jalr zero, 0(ra) # return via return address register
```

Calling a C Function from Assembly

```
.section .rodata
enter_prompt: .asciz "Enter a, b, and c: "
scan: .asciz "%d %d %d"
result_out: .asciz "Result = %d\n"
.section .text
.global main
main:
addi sp, sp, -32 # Allocate 32 bytes from the stack
sd ra, 0(sp) # Since we are making calls, we need the original ra
# Prompt the user first
la a0, enter_prompt
jal ra, printf
# We've printed the prompt, now wait for user input
```

```
la a0, scan
addi a1, sp, 8 # Address of a is sp + 8
addi a2, sp, 16 # Address of b is sp + 16
addi a3, sp, 24 # Address of c is sp + 24
jal ra, scanf
# Now all of the values are in memory, load them
# so we can jal ra, the c function.
lw a0, 8(sp)
lw a1, 16(sp)
lw a2, 24(sp)
jal ra, cfunc
# The result should be in a0, but that needs to be
# the second parameter to printf.
add a1, zero, a0
la a0, result_out
jal ra, printf
# Restore original RA and return
ld ra, 0(sp)
addi sp, sp, 32 # Always deallocate the stack!
jalr zero, 0(ra)
```

```
// extern "C" is required so we can link the name into assembly
// without knowing how C++ mangles it.
extern "C" {
int cfunc(int a, int b, int c);
}
// Simple function we're going to call from assembly.
int cfunc(int a, int b, int c) {
return a + b * c;
}
```

Binary Search

```
int binsearch(const int arr[], int needle, int size) {
int mid;
int left = 0;
```

```c
int right = size -1;
while (left <= right) {
mid = (left + right) / 2;
// Needle is bigger than what we're looking at,
// only consider the upper half of the list.
if (needle > arr[mid]) {
left = mid + 1;
}
else if (needle < arr[mid]) {
right = mid - 1;
}
else {
return mid;
}
}
// We've gone through all elements and the needle wasn't found.
return -1;
}
```

```asm
.section .text
.global binsearch
binsearch:
# a0 = int arr[]
# a1 = int needle
# a2 = int size
# t0 = mid
# t1 = left
# t2 = right
addi t1, zero, 0 # left = 0
addi t2, a2, -1 # right = size - 1
1: # while loop
bgt t1, t2, 1f # left > right, break
add t0, t1, t2 # mid = left + right
srai t0, t0, 1 # mid = (left + right) / 2
# Get the element at the midpoint
slli t4, t0, 2 # Scale the midpoint by 4
add t4, a0, t4 # Get the memory address of arr[mid]
```

```
lw t4, 0(t4) # Dereference arr[mid]
# See if the needle (a1) > arr[mid] (t3)
ble a1, t4, 2f # if needle <= t3, we need to check the next condition
# If we get here, then the needle is > arr[mid]
addi t1, t0, 1 # left = mid + 1
jal zero, 1b
2:
bge a1, t4, 2f # skip if needle >= arr[mid]
# If we get here, then needle < arr[mid]
addi t2, t0, -1 # right = mid - 1
jal zero, 1b
2:
# If we get here, then needle == arr[mid]
addi a0, t0, 0
1:
jalr zero, 0(ra)
```

Vector x Matrix

```
void matmul(float dst[3], float matrix[3][3], float vector[3]) {
int r;
int c;
float d;
for (r = 0;r < 3;r++) {
d = 0;
for (c = 0;c < 3;c++) {
d = d + matrix[r][c] * vector[c];
}
dst[r] = d;
}
}
```

We don't always have to translate 100% from C++ into assembly. We can take shortcuts if we know how the data structure is set up. matrix[3][3] can be calculated by using row * width + column, and then scaling it by the data size. However, we can take a shortcut and just advance the matrix pointer by each integer for every iteration of the loop. Since our outer and inner loops both iterate 3 times, we will iterate a total of 9 times–exactly the number of elements in our matrix.

```
.section .text
.global matmul
matmul:
# a0 = dst[3]
# a1 = matrix[3][3]
# a2 = vector[3]
# t0 = r
# t1 = c
# t3 = 3
# ft0 = d
# Row for loop
addi t0, zero, 0
addi t3, zero, 3
1:
bge t0, t3, 1f # break when we arere done
fmv.s.x fa0, zero # Set d = 0
# Column for loop
addi t1, zero, 0
2:
bge t1, t3, 2f
flw ft0, 0(a1) # Load matrix value
flw ft1, 0(a2) # Load vector value
fmul.s ft0, ft0, ft1 # ft0 = matrix[r][c] * vec[c]
fadd.s fa0, fa0, ft0 # d = d + ft0
addi t1, t1, 1
addi a1, a1, 4 # Move to the next matrix value
addi a2, a2, 4 # Move to the next vector value
jal zero, 2b
2:
addi a2, a2, -12 # Move the vector back to the top
fsw fa0, 0(a0) # dst[r] = d
addi t0, t0, 1
addi a0, a0, 4 # Move to next destination
jal zero, 1b
1:
jalr zero, 0(ra)
```