Tutorial2.1
Assembly language

——basic and gdb

Lei Wang

UNIVERSITY OF CALGARY

Basic instructions

- mov x1, x2 :Move register => x1 = x2
- add x1, x2, x3 : Add value => x1 = x2+x3
- bl printf :A function can be called by using the Branch and Link instruction => printf()
- ret: return to the caller
- .pseudo-op: assembler directives do not generate machine instructions, but give the assembler extra information

  Eg: .global start

- label: :a label can prefix any statement, is a symbol whose value is the address of the machine instruction

  Eg: start: add x20, x20, x21

Stack Frame

- Each Stack Frame corresponds to one function call
- **Stack Pointer**: the Stack Pointer(SP) always points to the top of the stack, if we want to call a function, we need to push a new Stack Frame to the stack, meaning we need to allocate more memory space
- **Frame Pointer**: The Frame Pointer(x29) is needed as an anchor, because SP can be be moved many times when saving variables, so we need a base address
- **Link Register**: The Link Register(x30) contains the return address for a function call.

Stack Frame

- stp x29, x30, [sp, -16]!: stores the content of the pair of registers to the stack, [sp, -16]! Allocates 16 bytes in stack memory(in RAM) by pre-incrementing the SP register by -16
- mov x29, sp: update FP to the current SP, FP may be used as a base address
- ldp x29, x30, [sp], 16: loads the pair of registers from RAM, restores the sate of the FP and LR registers, [sp], 16 Deallocates 16 bytes of stack memory by post-incrementing SP by +16

Editng, compiling and runing

- Edit: use vim

  ```
  vim hello.s
  ```

- Compile: use gcc

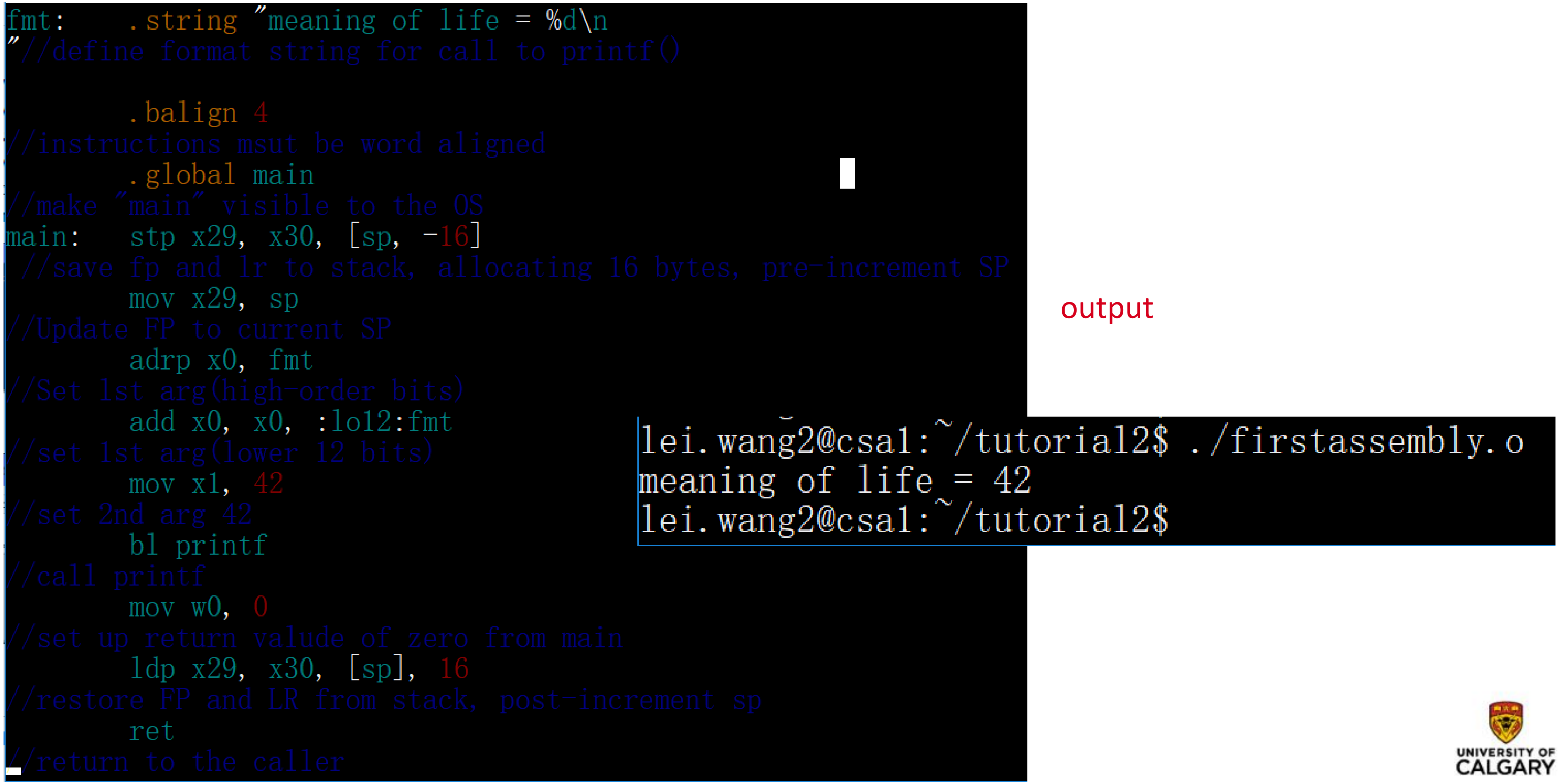
  ```
  gcc hello.s –o hello.out
  ```

  or gcc -g hello.s –o hello.out: connect debug information to code

- Run: ./hello.out

First assembly program!

- eg.



```
fmt:    .string "meaning of life = %d\n
"//define format string for call to printf()

        .balign 4
//instructions msut be word aligned
        .global main
//make "main" visible to the OS
main:   stp x29, x30, [sp, -16]
 //save fp and lr to stack, allocating 16 bytes, pre-increment SP
        mov x29, sp
//Update FP to current SP
        adrp x0, fmt
//Set 1st arg(high-order bits)
        add x0, x0, :lo12:fmt
//set 1st arg(lower 12 bits)
        mov x1, 42
//set 2nd arg 42
        bl printf
//call printf
        mov w0, 0
//set up return valude of zero from main
        ldp x29, x30, [sp], 16
//restore FP and LR from stack, post-increment sp
        ret
//return to the caller
```

output

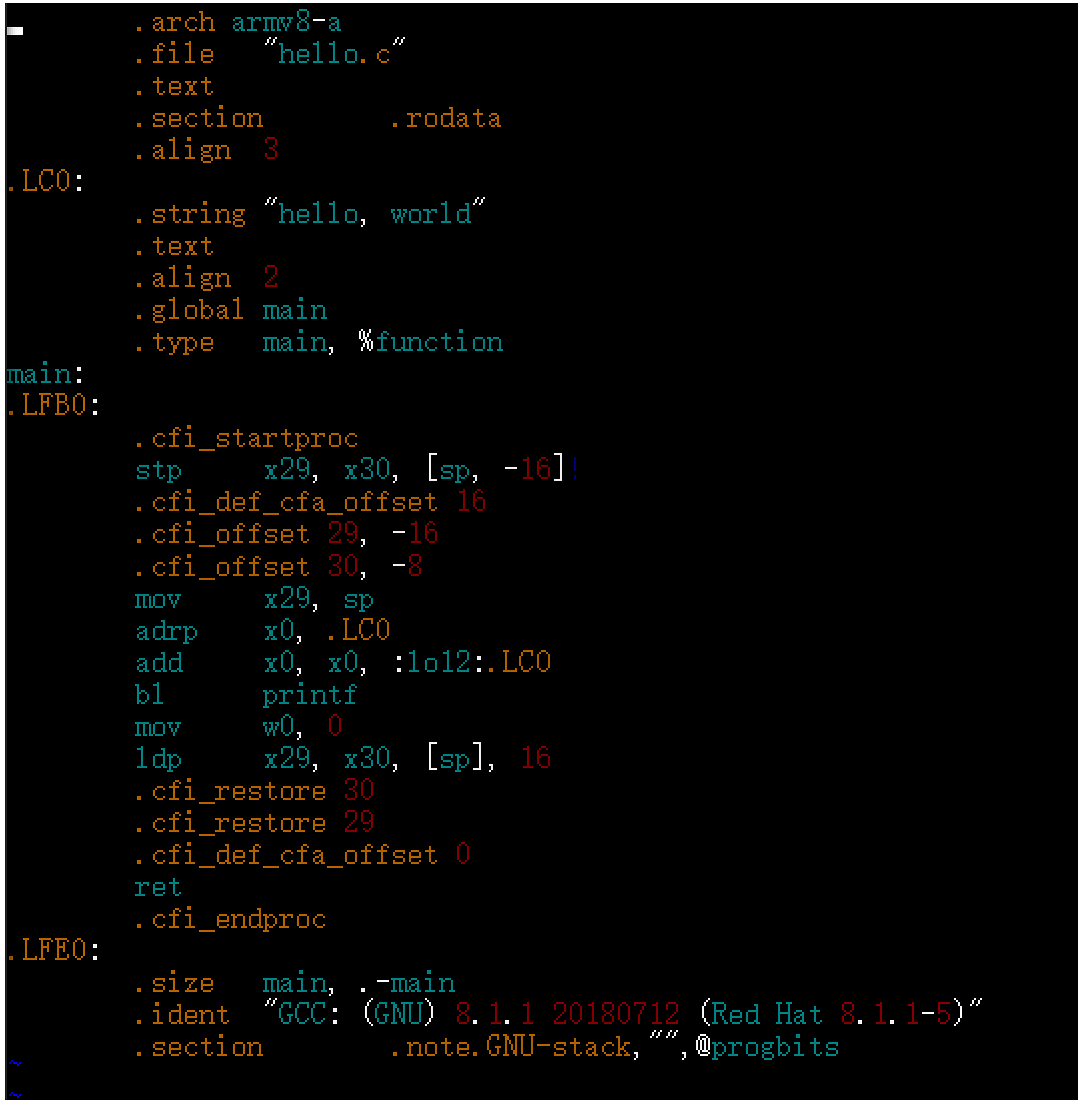
```
lei.wang2@csa1:~/tutorial2$ ./firstassembly.o
meaning of life = 42
lei.wang2@csa1:~/tutorial2$
```

Get an assembly file from c file

- Eg.

  gcc –S hello.c –o hello.s

  vim hello.s



```
        .arch armv8-a
        .file   "hello.c"
        .text
        .section        .rodata
        .align  3
.LC0:
        .string "hello, world"
        .text
        .align  2
        .global main
        .type   main, %function
main:
.LFB0:
        .cfi_startproc
        stp     x29, x30, [sp, -16]!
        .cfi_def_cfa_offset 16
        .cfi_offset 29, -16
        .cfi_offset 30, -8
        mov     x29, sp
        adrp    x0, .LC0
        add     x0, x0, :lo12:.LC0
        bl      printf
        mov     w0, 0
        ldp     x29, x30, [sp], 16
        .cfi_restore 30
        .cfi_restore 29
        .cfi_def_cfa_offset 0
        ret
        .cfi_endproc
.LFE0:
        .size   main, .-main
        .ident  "GCC: (GNU) 8.1.1 20180712 (Red Hat 8.1.1-5)"
        .section        .note.GNU-stack,"",@progbits
~
~
```

Debug—using gdb

- gdb [excutable filename]: starting gdb
- r: run a program
- b [line number]/[function name]: setting breakpoint
- c: continuing after a break
- ni and si: execute the next instruction, ni proceeds until the function returns
- list : show the source code
- p $reg: print the contents of a resgiter
- x and x/[count][format] [expression]: examining memory
- display/i $pc: show the current instruction when single step
- layout src: show source code and the current running instruction
- q: quiting gdb

Macros

- Pre-defined compiler :allows you to define a piece of text with a macro name

eg:

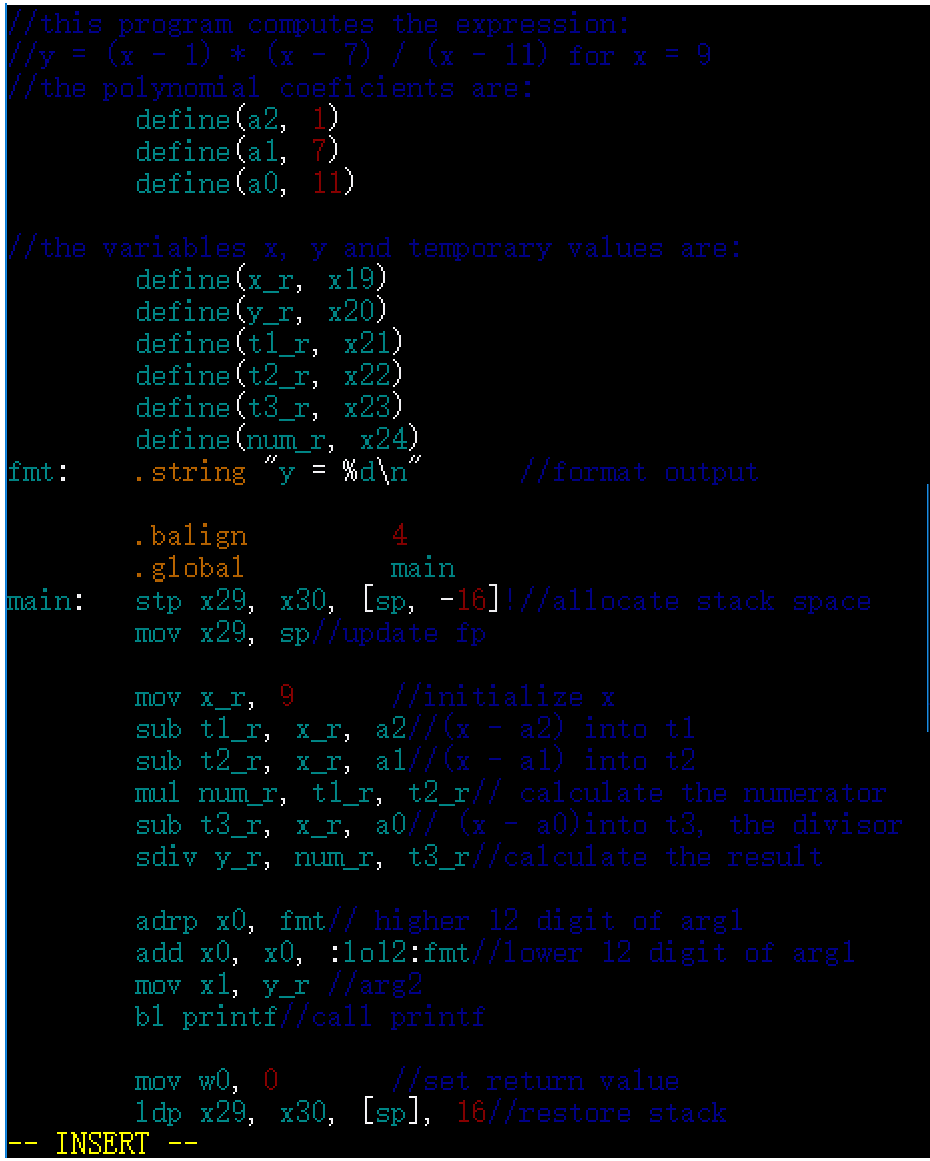
```
define(coef, 23)
define(z_f, x18)
        …
        add x19, z_r, coef
        …
Is expanded to:
        …
        add x19, x18, 23
        …
```

Use Macros:

```
vim hello.asm
m4 hello.asm > hello.s
gcc –g hello.s –o hello.out
```

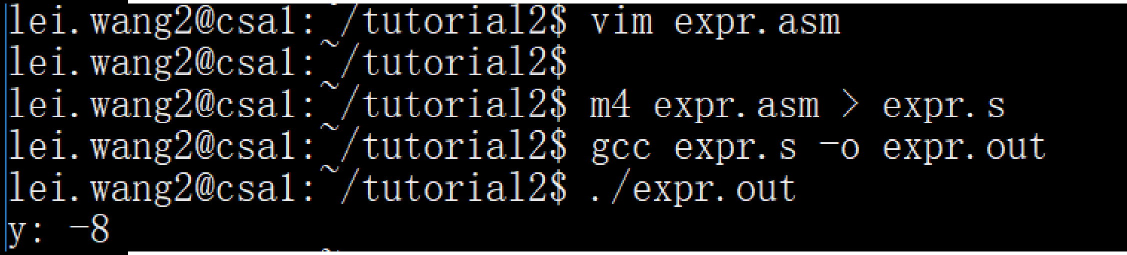
Macros

- example



```
//this program computes the expression:
//y = (x - 1) * (x - 7) / (x - 11) for x = 9
//the polynomial coeficients are:
        define(a2, 1)
        define(a1, 7)
        define(a0, 11)

//the variables x, y and temporary values are:
        define(x_r, x19)
        define(y_r, x20)
        define(t1_r, x21)
        define(t2_r, x22)
        define(t3_r, x23)
        define(num_r, x24)
fmt:    .string “y = %d\n”      //format output

        .balign         4
        .global         main
main:   stp x29, x30, [sp, -16]!//allocate stack space
        mov x29, sp//update fp

        mov x_r, 9      //initialize x
        sub t1_r, x_r, a2//(x - a2) into t1
        sub t2_r, x_r, a1//(x - a1) into t2
        mul num_r, t1_r, t2_r// calculate the numerator
        sub t3_r, x_r, a0// (x - a0)into t3, the divisor
        sdiv y_r, num_r, t3_r//calculate the result

        adrp x0, fmt// higher 12 digit of arg1
        add x0, x0, :lo12:fmt//lower 12 digit of arg1
        mov x1, y_r //arg2
        bl printf//call printf

        mov w0, 0       //set return value
        ldp x29, x30, [sp], 16//restore stack
-- INSERT --
```

calculate (x-1)*(x-7)/(x-11) for x=9



```
lei.wang2@csa1:~/tutorial2$ vim expr.asm
lei.wang2@csa1:~/tutorial2$
lei.wang2@csa1:~/tutorial2$ m4 expr.asm > expr.s
lei.wang2@csa1:~/tutorial2$ gcc expr.s -o expr.out
lei.wang2@csa1:~/tutorial2$ ./expr.out
y: -8
```

exercise

- Try using gdb to debug the macros example before
- Practice all the command in gdb