

Computer Organization

Lab 1: RISC-V Programming

郭家宏

Lab 1 : RISC-V Assembly Programming

- Factorial
- Bubble_sort
- Gcd
- Fibonacci

| Register | ABI Name | Description | Saver |
|----------|----------|-----------------------------------|--------|
| x0 | zero | Hard-wired zero | — |
| x1 | ra | Return address | Caller |
| x2 | sp | Stack pointer | Callee |
| x3 | gp | Global pointer | — |
| x4 | tp | Thread pointer | — |
| x5 | t0 | Temporary/alternate link register | Caller |
| x6–7 | t1–2 | Temporaries | Caller |
| x8 | s0/fp | Saved register/frame pointer | Callee |
| x9 | s1 | Saved register | Callee |
| x10–11 | a0–1 | Function arguments/return values | Caller |
| x12–17 | a2–7 | Function arguments | Caller |
| x18–27 | s2–11 | Saved registers | Callee |
| x28–31 | t3–6 | Temporaries | Caller |
| f0–7 | ft0–7 | FP temporaries | Caller |
| f8–9 | fs0–1 | FP saved registers | Callee |
| f10–11 | fa0–1 | FP arguments/return values | Caller |
| f12–17 | fa2–7 | FP arguments | Caller |
| f18–27 | fs2–11 | FP saved registers | Callee |
| f28–31 | ft8–11 | FP temporaries | Caller |

Table: System services.

| Service | System Call Code | Arguments | Result |
|-----------------|------------------|--|---------------------------|
| print_int | 1 | \$a0 = integer | |
| print_float | 2 | \$f12 = float | |
| print_double | 3 | \$f12 = double | |
| print_string | 4 | \$a0 = string | |
| read_int | 5 | | integer (in \$v0) |
| read_float | 6 | | float (in \$f0) |
| read_double | 7 | | double (in \$f0) |
| read_string | 8 | \$a0 = buffer, \$a1 = length | |
| sbrk | 9 | \$a0 = amount | address (in \$v0) |
| exit | 10 | | |
| print_character | 11 | \$a0 = character | |
| read_character | 12 | | character (in \$v0) |
| open | 13 | \$a0 = filename, \$a1 = flags, \$a2 = mode | file descriptor (in \$v0) |
| read | 14 | \$a0 = file descriptor, \$a1 = buffer, \$a2 = count | bytes read (in \$v0) |
| write | 15 | \$a0 = file descriptor, \$a1 = buffer, \$a2 = count | bytes written (in \$v0) |
| close | 16 | \$a0 = file descriptor | 0 (in \$v0) |
| exit2 | 17 | \$a0 = value | |

Table 20.1: Assembler mnemonics for RISC-V integer and floating-point registers.

Revision

Register Operand Example

- C code:

```
f = (g + h) - (i + j);
```

– f, ..., j in x19, x20, ..., x23

- Compiled RISC-V code:

```
add x5, x20, x21  
add x6, x22, x23  
sub x19, x5, x6
```

Revision

Memory Operand Example

- ❑ C code:

`A[12] = h + A[8];`

- h in x21, base address of A in x22

- ❑ Compiled RISC-V code:

- Index 8 requires offset of 64

- ❑ 8 bytes per doubleword

| | |
|-----|-------------|
| ld | x9, 64(x22) |
| add | x9, x21, x9 |
| sd | x9, 96(x22) |

Revision

Immediate Operands

- ❑ Constant data specified in an instruction
`addi x22, x22, 4`
- ❑ **Design Principle 3:** Make the common case fast
 - Small constants are common
 - Immediate operand avoids a load instruction

Revision

Compiling If Statements

❑ C code:

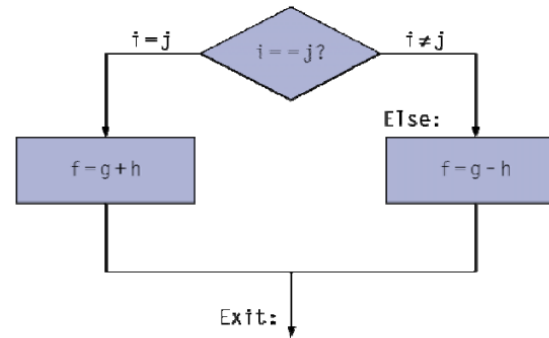
```
if (i==j) f = g+h;  
else f = g-h;
```

– f, g, ... in x19, x20, ...

❑ Compiled RISC-V code:

```
    bne x22, x23, Else  
    add x19, x20, x21  
    beq x0,x0,Exit // unconditional  
Else: sub x19, x20, x21  
Exit: ...
```

Assembler calculates addresses



Revision Example

Leaf Procedure Example

❑ RISC-V code:

fact:

addi sp,sp,-16

sd x1,8(sp)

sd x10,0(sp)

addi x5,x10,-1

bge x5,x0,L1

addi x10,x0,1

addi sp,sp,16

jalr x0,0(x1)

L1: addi x10,x10,-1

jal x1,fact

addi x6,x10,0

ld x10,0(sp)

ld x1,8(sp)

addi sp,sp,16

mul x10,x10,x6

jalr x0,0(x1)

Save return address and n on stack

$x5 = n - 1$

if $n \geq 1$, go to L1

Else, set return value to 1

Pop stack, don't bother restoring values

Return

$n = n - 1$

call fact(n-1)

move result of fact(n - 1) to x6

Restore caller's n

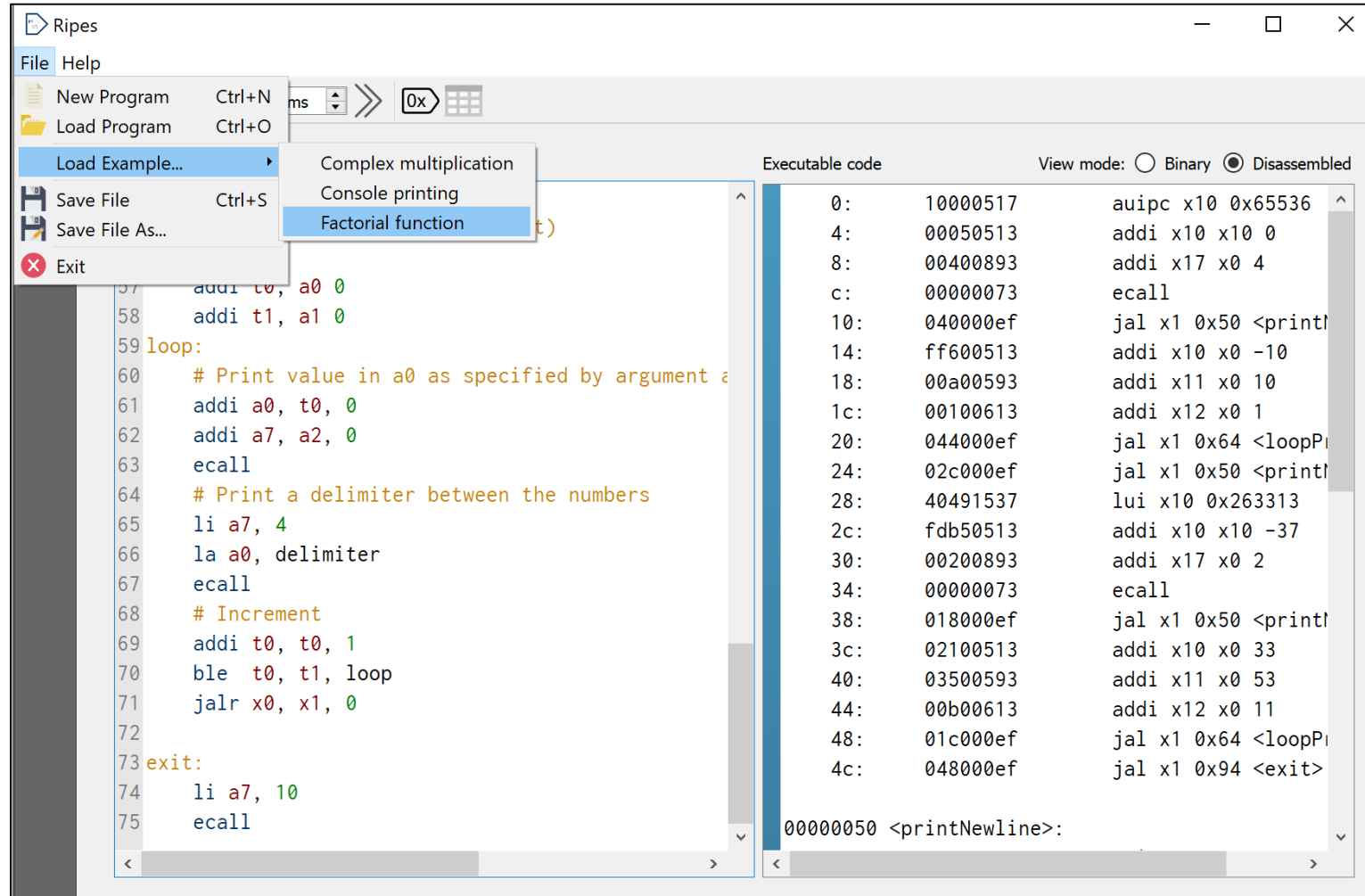
Restore caller's return address

Pop stack

return $n * \text{fact}(n-1)$

return

Example



Ripes

File Help

New Program Ctrl+N

Load Program Ctrl+O

Load Example...

Save File Ctrl+S

Save File As...

Exit

Complex multiplication

Console printing

Factorial function

ms >> 0x

Executable code View mode: ☐ Binary ☒ Disassembled

```
57 addi t0, a0, 0
58 addi t1, a1, 0
59 loop:
60 # Print value in a0 as specified by argument a0
61 addi a0, t0, 0
62 addi a7, a2, 0
63 ecall
64 # Print a delimiter between the numbers
65 li a7, 4
66 la a0, delimiter
67 ecall
68 # Increment
69 addi t0, t0, 1
70 ble t0, t1, loop
71 jalr x0, x1, 0
72
73 exit:
74 li a7, 10
75 ecall
```

0: 10000517 auipc x10 0x65536
4: 00050513 addi x10 x10 0
8: 00400893 addi x17 x0 4
c: 00000073 ecall
10: 040000ef jal x1 0x50 <printNewline>
14: ff600513 addi x10 x0 -10
18: 00a00593 addi x11 x0 10
1c: 00100613 addi x12 x0 1
20: 044000ef jal x1 0x64 <loopPrint>
24: 02c000ef jal x1 0x50 <printNewline>
28: 40491537 lui x10 0x263313
2c: fdb50513 addi x10 x10 -37
30: 00200893 addi x17 x0 2
34: 00000073 ecall
38: 018000ef jal x1 0x50 <printNewline>
3c: 02100513 addi x10 x0 33
40: 03500593 addi x11 x0 53
44: 00b00613 addi x12 x0 11
48: 01c000ef jal x1 0x64 <loopPrint>
4c: 048000ef jal x1 0x94 <exit>

00000050 <printNewline>

How to calculate the instruction

Example: factorial.s

- In this case, there are 121 instructions will be executed.
- There are 16 variables in stack at most.

```
main:
    lw      a0, argument      1
    jal     ra, fact          2

    # Print the result to console
    mv      a1, a0            102
    lw      a0, argument      103
    jal     ra, printResult    104

    # Exit program
    li      a0, 10            120
    ecall                               121

fact:
    addi     sp, sp, -16      3      10      17      24      31      38      45      52
    sw       ra, 8(sp)        4      11      18      25      32      39      46      53
    sw       a0, 0(sp)        5      12      19      26      33      40      47      54
    addi     t0, a0, -1       6      13      20      27      34      41      48      55
    bge      t0, zero, nfact  7      14      21      28      35      42      49      56

    addi     a0, zero, 1      57
    addi     sp, sp, 16       58
    jalr     x0, x1, 0        59

nfact:
    addi     a0, a0, -1       8      15      22      29      36      43      50
    jal      ra, fact         9      16      23      30      37      44      51

    addi     t1, a0, 0        96      90      84      78      72      66      60
    lw       a0, 0(sp)        97      91      85      79      73      67      61
    lw       ra, 8(sp)        98      92      86      80      74      68      62
    addi     sp, sp, 16       99      93      87      81      75      69      63

    mul      a0, a0, t1       100     94      88      82      76      70      64
    ret                               101     95      89      83      77      71      65
    #jalr x0, ra, 0
```

Demo

