Discussion 4 RISC-V

- yangchao@shanghaitech.edu.cn

Agenda

上海科技大学
ShanghaiTech University

1.RISC-V Calling Convention

2.Jump

3.Label and Assembler Directives

4.Enviroment calls

5.Exam examples

立志成才报国裕民

① exam

浮点数加法
浮点 ⟺ int
risc-V add x1, x2, x3 ⇒ 二进制.

C语言
risc-V 汇编 ⟨ 阅读 / 填空 *

② CA

CPU

算法 → (代码. ⟶ 硬件

C语言
java
python

C语言 转化成 基本指令

printf scanf return 0; ⟹ ecall;

① 算术 + - * / ⟸ c=a+b; i=i+1;

② if, while, for,

③ 调用 jump jal jalr.

risc-V 汇编 ⟹ 数字电路

流水线 ⟹ { 算 ← / 存 ←

cache.
一级 二级

C语言 → 汇编 → 硬件

open MP

void swap (long long int v[], int k) DH Chapter 2.13

a0 a7

x10 x17

```
{
  long long int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

v[k] address.

x6 = k*8.

```
  slli  x6, x11, 3
  add   x6, x10, x6.      v[k].
8 ld    x5, 0(x6)
  ld    x7, 8(x6)         v[k+1]
8 sd    x7, 0(x6)
  sd    x5, 8(x6)
```

```
void sort (long long int v[], int n)
{
    int i,j;
    for (i=0; i<n; i+=1) {
      for (j=i-1; j>=0 && v[j]>v[j+1]; j-=1) {
          swap(v,j);
      }
    }
}
```

冒泡

i>=n

※ RV64

一个 register

64 bit

与课上32个

bit 不一样!

考试, project, HW

以RV32为准!

x10 , x11

for (i=0; i<n; i=i+1) { ... }

x19 ⇒ i

x20 ⇒ j

① 赋值

② 判断

- - - -

③ i=i+1

(i>=n)

```
      addi  x19, x0, 0        if x19>=x11   exit;
for1: bge   x19, x11, exit1
      ...
      addi  x19, x19, 1.
      j     for1
```

for (j=i-1; j>=0 && v[j]>v[j+1]; j-=1)

j=i-1 ⇒ addi x20, x19, -1 ← 赋值

```
for2:
    blt   x20, 0, exit2     // if (j<0) exit
    slli  x5, x20, 3        (j*8) ⇒ x5.
    add   x5, x10, x5       reg x5 = v+(j*8)
    ld    x6, 0(x5)         reg x6 = v[j]
    ld    x7, 8(x5)
    ble   x6, x7, exit2.
    ...
    addi  x20, x20, -1
    j     for2
```

判断

swap (v, j)

x10, x11 的值复制到 x21, x22

x1, x22, x21, x20, x19

```
addi  sp, sp, -40.
sd    x1, 32(sp).
sd    x22, 24(sp)
sd    x21, 16(sp)
sd    x20, 8 (sp)
sd    x19, 0 (sp)
```

RISC-V Calling Convention

- The "Calling Convention" in the ABI is the format/usage of registers in a way between the function ***caller*** and function ***callee***, if all functions implement it, everything works out
  - It is effectively a contract between functions
- Registers are two types:
  - ***caller-saved:*** The function invoked (the callee) can do whatever it wants to them!
  - ***callee-saved:*** The function invoked must restore them before returning (if used).

访问速度 & 空间大小

- Values saved by the **caller** before jumping to a function using **jal**
  - ra: Return address, used in function call.
  - a0-a1: Function argument and return values, also argument of environment call.
  - a2-a7: Function argument, used to pass parameters in function call.
  - t0-t6: Temporaries, cannot trust them after function call.

- Values restored by the **callee** before returning from a function using **jalr**
  - sp: Stack pointer. We subtract from sp to create more space and add to free space.

The stack is mainly used to save (and later restore) the value of registers that may be overwritten.

  - s0-s11: Saved registers, should not change after function call.

Caller & Callee

- Caller invoke callee
- Caller should save Caller-Saved registers (to memory) before the call.
- Callee should save Callee-Saved registers at the beginning of its execution and restore them before return.

caller 将一些 caller saved register 保存到栈上

Steps of function call :

1. Caller put parameters into registers a0-a7.
2. Caller put next line' s address into ra and jump to the function label. (using jal)
3. Callee pushes s0-s11, sp onto stack. callee saved 存.
4. Callee execution.
5. Callee extract value from stack. callee saved 恢复
6. Callee jump to ra' s address.

caller 将 caller saved register 恢复.

REGISTER NAME, USE, CALLING CONVENTION

④

REGISTER	NAME	USE	SAVER
x0	zero	The constant value 0	N.A.
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	--
x4	tp	Thread pointer	--
x5-x7	t0-t2	Temporaries	Caller
x8	s0/fp	Saved register/Frame pointer	Callee
x9	s1	Saved register	Callee
x10-x11	a0-a1	Function arguments/Return values	Caller
x12-x17	a2-a7	Function arguments	Caller
x18-x27	s2-s11	Saved registers	Callee
x28-x31	t3-t6	Temporaries	Caller

```
sum_squares:

prologue:
    addi sp sp -16
    sw s0 0(sp)
    sw s1 4(sp)
    sw s2 8(sp)
    sw ra 12(sp)
    li s0 1
    mv s1 a0
    mv s2 0

loop_start:
    bge s0 s1 loop_end
    mv a0 s0
    jal square
    add s2 s2 a0
    addi s0 s0 1
    j loop_start

loop_end:
    mv a0 s2

epilogue:
    lw s0 0(sp)
    lw s1 4(sp)
    lw s2 8(sp)
    lw ra 12(sp)
    addi sp sp 16
    jr ra
```

callee saved

ra
caller saved

sum_squares 作为callee

作为caller

恢复.

```c
void QuickSort(int *arr,int low,int high){
        if(low<high){
        int i=low;
        int j=high;
        int key=arr[low];
        while(i<j){
            while(i<j&&arr[j]>=key)
                j--;
            if(i<j)
                arr[i++]=arr[j];
            while(i<j&&arr[i]<key)
                i++;
            if(i<j)
                arr[j--]=arr[i];
        }
        arr[i]=key;
        QuickSort(arr,low,i-1);
        QuickSort(arr,i+1,high);
        }
    }

int main(){
        int a[11]={25,1,12,25,10,10,34,900,23,12,80};
        QuickSort(a,0,10);
        int i;
        for(i=0;i<11;i++){
            printf("%d ",a[i]);
        }
        return 0;
    }
```

Jump

- j指令操作为PC-relative Jump，它将PC 设置为 **PC + offset**，它实际上为pseudo-insturction，它使用的JAL接口，但将rd设置为x0（即忽略返回地址）。

- jr指令虽然也用于转移，但它是I类型，它**并不是** PC-relative Jump，它将PC 设置为 **rs + offset**，因为它提供的是一个寄存器，由于寄存器中存储的就是32位的数据，所以指令无需进行指令那样的取位操作，它实际上也是pseudo-insturction， 它使用的JALR接口，但将rd设置为x0（即忽略返回地址）。

- jal与j指令的唯一区别是在跳转之前，将下一条指令的PC地址赋值给$ra寄存器，j是一种无条件跳转的指令，而jal则是以比较明显的方法声明子程序调用的结构。

- j/jr与jal/jalr关系类似，左右两边都分别代表使用立即数或寄存器，同样地，jalr也是I类型的指令。

Generally,
when the main function calls other functions, use **jal ra,function**
when some function return, use **jr ra**
when in a loop, iterate by **jal x0,loop**

jr rs == jalr x0,rs

j offset == jal x0,offset

jal offset == jal ra,offset

Label and Assembler Directives

Label : Hold the address of data or instructions.
(Will be placed by the actual address during assembly or link.)

Some directives :

Directive	Effect
`.data`	Store subsequent items in the static segment at the next available address.
`.text`	Store subsequent instructions in the text segment at the next available address.
`.byte`	Store listed values as 8-bit bytes.
`.asciiz`	Store subsequent string in the data segment and add null-terminator.
`.word`	Store listed values as unaligned 32-bit words.
`.globl`	Makes the given label global.
`.float`	Reserved.
`.double`	Reserved.
`.align`	Reserved.

static

字符串

伪

risc-v
程序

```
.data
course:
    .asciiz "cs110"
semester:
    .asciiz "sp21"
num:
    .word 2021

.text
    la a1, course
    addi a0, x0, 4      # ecall 4 -- print_string
    ecall

    addi a1, x0, 10     # ASCII 10 -- '\n'
    addi a0, x0, 11     # ecall 11 -- print_character
    ecall

    la a1, semester
    addi a0, x0, 4      # ecall 4 -- print_string
    ecall

    addi a0, x0, 10     # ecall 10 -- exit
    ecall
```

Output :

```
cs110
sp21
```

Ecall

To use an environmental call, load the ID into register `a0`, and load any arguments into `a1` - `a7`. Any return values will be stored in argument registers.

The following environmental calls are currently supported.

ID (`a0`)	Name	Description
1	print_int ✓	prints integer in `a1`
4	print_string ✓	prints the null-terminated string whose address is in `a1`
9	sbrk	allocates `a1` bytes on the heap, returns pointer to start in `a0`
10	exit ✓	ends the program
11	print_character ✓	prints ASCII character in `a1`
13	openFile	Opens the file in the VFS where a pointer to the path is in `a1` and the permission bits are in `a2`. Returns to a0 an integer representing the file descriptor.
14	readFile	Takes in: `a1` = FileDescriptor, `a2` = Where to store the data (an array), `a3` = the amount you want to read from the file. Returns `a0` = Number of items which were read and put to the given array. If it is less than `a3` it is either an error or EOF. You will have to use another ecall to determine what was the cause.
15	writeFile	Takes in: `a1` = FileDescriptor, `a2` = Buffer to read data from, `a3` = amount of the buffer you want to read, `a4` = Size of each item. Returns `a0` = Number of items written. If it is less than `a3` it is either an error or EOF. You will have to use another ecall to determine what was the cause. Also, you need to flush or close the file for the changes to be written to the VFS.

16	closeFile	Takes in: `a1` = FileDescriptor. Returns 0 on success and EOF (-1) otherwise. Will flush the data as well.
17	exit2	ends the program with return code in `a1`
18	fflush	Takes in: `a1` = FileDescriptor. Will return 0 on success otherwise EOF on an error.
19	feof	Takes in: `a1` = FileDescriptor. Returns a nonzero value when the end of file is reached otherwise, it returns 0.
20	ferror	Takes in: `a1` = FileDescriptor. Returns Nnnzero value if the file stream has errors occurred, 0 otherwise.
34	printHex	prints hex in `a1`
0x3CC	vlib	Please check out the vlib page to see what functions you can use!

The environmental calls are intended to be somewhat backwards compatible with SPIM's syscalls.

As an example, the following code prints the integer `42` to the console:

```
addi a0 x0 1        # print_int ecall
addi a1 x0 42       # integer 42
ecall
```

Exam examples

上海科技大学
ShanghaiTech University

Perform an R-type **signed** addition (add t2, t1, t0) and detect overflow. If an overflow occurs, t4 register is set to 1; otherwise, t4 is set to 0. Please use RV32I instructions (as less instructions as possible) to complete the below assembly code. (**Hint**: the sum should be less than one of the operands if and only if the other operand is negative. Feel free to use `t0~t6`. Also, please comment your code properly. Leave it blank if you do not use all the space below.)

```
add   t2, t1, t0

________________

________________

________________

________________

________________
addi  t3, t3, 1
beq   t4, t3, OVERFLOW
... ...
OVERFLOW: #some code to deal with overflow
```

Exam examples

Perform an R-type **signed** addition (add t2, t1, t0) and detect overflow. If an overflow occurs, t4 register is set to 1; otherwise, t4 is set to 0. Please use RV32I instructions (as less instructions as possible) to complete the below assembly code. (**Hint**: the sum should be less than one of the operands if and only if the other operand is negative. Feel free to use `t0~t6`. Also, please comment your code properly. Leave it blank if you do not use all the space below.)

```
add   t2, t1, t0
______________
______________
______________
______________
______________
addi  t3, t3, 1
beq   t4, t3, OVERFLOW
... ...
OVERFLOW: #some code to deal with overflow
```

$t0 \geq 0$
$t2 \geq t1$
$t1 + t0 \geq t1$
$t2 < t1$

$t0 < 0$
$t1 + t0 < t1$
$t2 >= t1$

Solution:

```
slti t5, t0, 0 #set t5 if t0<0
slt t4, t2, t1 #set t4 if t2<t1
xor t4, t4, t5 #if (t0>=0 and t2<t1) or (t0<0 and t2>=t1),
    overflow occurs and t4 is set to 1
addi t3, x0, 0 #initialize t3.
```

t1 and t2 are exchangable. t5 can be other tx.

Thank you for attending the discussion!

Wish you good luck doing homework/projects/exams!