香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab 1-2: RISC-V Assembly Language Programing II

Chen BAI
Department of Computer Science & Engineering
Chinese University of Hong Kong
cbai@cse.cuhk.edu.hk

Spring 2022

Outline

Recap

Categories

- Load and Store Instructions
- Bitwise Instructions
- Arithmetic Instructions
- Control Transfer Instructions

Load and Store Instructions

- Load and store instructions: `lb lbu lh lhu lw sb sh sw`
- Immediate instructions: `lui auipc`
- Pseudo instructions: `mv li la`

Bitwise Instructions

- **Register to register instructions:** `sll srl sra and or xor not slt sltu`
- **Immediate instructions:** `slli srli srai andi ori xori slti sltiu`

Arithmetic Instructions

- Register to register instructions: `add sub mul mulh mulhu mulhsu div divu rem`
- Immediate instructions: `addi`

Control Transfer Instructions

- **Branch instructions:** `beq bne blt bltu bge bgeu`
- **Unconditional jump instructions:** `jal jalr ret j jr`
- **Pseudo instructions:** `beqz bnez blez bgez bltz bgtz bgt ble bgtu bleu`

RISC-V Assembler Directives

- **Object file section:** `.text` `.data` `.rodata` `.bss` `.comm` `.common` `.section`
- **Misc. functions:** `.option` `.file` `.ident` `.size` `.type`
- **Definition and exporting of symbols:** `.globl` `.local` `.equ`
- **Alignment control:** `.align` `.balign` `.p2align`
- **Emitting data:** `.byte` `.2byte` `.4byte` `.8byte` `.half` `.word` `.dword` `.asciz` `.string` `.incbin` `.zero`

Declaration

```
.data
a:   .word 1 2 3 4 5
```

Remark

- Similar to the definition of array in C++, "`a`" denotes the address of the first element of the array.
- We can access through rest of the elements with *.word* offset (*i.e.*, 4 bytes). (What should be the offset for the 2^{nd} element in the array above?)

Examples I

Example 1

```
_start:
    addi t0, t0, 0
    addi t1, t1, 0
    andi t2, t2, 0
    li t0, 0xFF          # Load a 8-bit number
    li t1, 0xFFFF        # Load a 32-bit number
    li t2, 0xFFFFFFFF    # Load a 64-bit number
```

Example 2

```
_start:
    addi t0, t0, 0
    addi t1, t1, 0
    andi t2, t2, 0
    li t0, 0x1A352A9C    # t0 = 0x1A352A9C
    li t1, 0x1B2D4C6A    # t1 = 0x1B2D4C6A
    addi t2, t0, t1      # t2 = t1 + t0
```

Examples III

Example 3

Examples IV

```
_start:
    addi t0, t0, 0
    addi t1, t1, 0
    andi t2, t2, 0
    andi t3, t3, 0
    andi t4, t4, 0
    andi t5, t5, 0
    li t0, 2                    # t0 = 2
    li t3, -2                   # t3 = -2
    slt t1, t0, zero            # t1 = 1 if t0 < 0
    beq t1, zero, else_if
    j end_if
else_if:
    sgt t4, t3, zero            # t4 = 1 if t3 > 0
    beq t4, zero, else
    j end_if
else:
    seqz t5, t4, zero           # t5 = 1 if t4 = 0
end_if:
    j end_if
```

Function Call Procedure

Function Call Procedure

JAL

- The JAL instruction is used to call a subroutine (*i.e.*, function).
- The return address (*i.e.*, the PC, which is the address of the instruction following the JAL) is saved in the destination register.
- The target address is given as a PC-relative offset (the offset is sign-extended, multiplied by 2, and added to the value of the PC).

Function Call Procedure I

Syntax

jal rd, offset

Usage

```
loop: addi x5, x4, 1      # assign x4 + 1 to x5
jal x1, loop              # assign 'PC + 4' to x1 and jump to loop
```

Function Call Procedure

JALR

- The JALR instruction is used to call a subroutine (*i.e.*, function).
- The return address (*i.e.*, the PC, which is the address of the instruction following the JALR) is saved in the destination register.
- The target address is given as a PC-relative offset (the offset is sign-extended and added to the value of the destination register).

Function Call Procedure I

Syntax

jal rd, offset

Usage

```
addi x1, x0, 3          # assign x0 + 3 to x1
loop: addi x5, x0, 1    # assign x0 + 1 to x5
jalr x0, 64(x1)         # assign 'PC + 4' to x1 and jump to address x1
     + 64
```

Function Call Procedure
Difference between JAL & JALR

31	30 ... 21	20	19 ... 12	11 ... 7	6 ... 0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	10	1	8	5	7
offset[20:1]				dest	JAL

31 ... 20	19 ... 15	14 ... 12	11 ... 7	6 ... 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	0	dest	JALR

More Examples of Function Call Procedure I

J

A pseudo instruction for JAL

Syntax

j label

Usage

```
loop: addi x5, x4, 1     # assign x4 + 1 to x5
jal loop                 # assign 'PC + 4' to x0 and jump to loop (
    discard the return address)
```

More Examples of Function Call Procedure II

JR

A pseudo instruction for JALR

Syntax

jr rs1

Usage

```
label: li x28, 100       # assign 100 to x28
li x5, 200               # assign 200 to x5
li x6, 50                # assign 50 to x6
jal ra, loop             # jump to loop
li x2, 10                # assign 10 to x2
loop: add x4, x28, x5    # assign x28 + x5 to x4
sub x7, x6, x4           # assign x6 + x4 to x7
jr ra                    # jump to 'ra + 0'
```

More Examples of Function Call Procedure III

BEQ

If the values stored in `rs1` and `rs2` are equal, jump to `label`.

Syntax

beq rs1, rs2, label

Usage

```
beq x1, x0, loop # jump to loop when x1 equals to 0
```

Remark

It is similar to `bne, blt, bltu, bge, bgeu`...

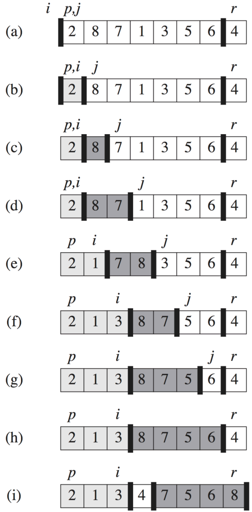
Array Partitioning

Partitioning

- Pick an element, called a pivot, from the array.
- Reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way).

```
function PARTITION(A, lo, hi)
    pivot ← A[hi]
    i ← lo-1;
    for j = lo; j ≤ hi-1; j ← j+1 do
        if A[j] ≤ pivot then
            i ← i+1;
            swap A[i] with A[j];
        end if
    end for
    swap A[i+1] with A[hi];
    return i+1;
end function
```

Example of Partition



[1]

---

[1] In this example, p = lo and r = hi.

Lab 1-2 Assignment

Lab Assignment

An array `array1` contains the sequence `-1 22 8 35 5 4 11 2 1 78`, each element of which is *.word*. Rearrange the element order in this array such that,

1. All the elements smaller than the 3^{rd} element (i.e. 8) are on the left of it,
2. All the elements bigger than the 3^{rd} element (i.e. 8) are on the right of it.

Submission Method:

Submit the source code and report after the whole lectures of Lab1 into **Blackboard**.

- The report includes your thinking, illustration of implementations, and results of the source code.

Swap v[k] and v[k+1]

Assume `a0` stores the address of the first element and `a1` stores k.

```
swap: sll t1, a1, 2      # get the offset of v[k] relative
    to v[0]
      add t1, a0, t1     # get the address of v[k]
      lw  t0, 0(t1)      # load the v[k] to t0
      lw  t2, 4(t1)      # load the v[k + 1] to t2
      sw  t2, 0(t1)      # store t2 to the v[k]
      sw  t0, 4(t1)      # store t0 to the v[k + 1]
```

Appendix-B Simple Sort Example

C style sort:

```
void sort(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(j + 1, j);
        }
    }
}
```

Appendix-C Save and Exit

Exit and restoring registers

```
exit1:
    lw   ra, 16(sp)
    lw   s3, 12(sp)
    lw   s2, 8(sp)
    lw   s1, 4(sp)
    lw   s0, 0(sp)
    addi sp, sp, 20
```