CENG3420

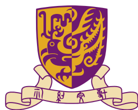
Lab 1-2: Control Instructions and Function Call

Chen BAI

Department of Computer Science and Engineering
The Chinese University of Hong Kong

cbai@cse.cuhk.edu.hk

Spring 2021



香港中文大學
The Chinese University of Hong Kong

Recap I

Load and store instructions

- `lb lbu lh lhu lw sb sh sw lui`
- `ld sd lwu`

Bitwise instructions

- `sll srl sra or xor not slt sltu slli srli srai andi ori xori slti sltiu`
- `sllw srlw sraw srliw sraiw`

Arithmetic instructions

- `add sub mul mulh mulhu mulhsu div divu rem`
- `addi addw subw remu mulw divw divuw remw remuw addiw`

Recap II

Control transfer instructions

- `beq bne blt bltu bge bgeu`
- `ret jal jalr`

Pseudo instructions

- `beqz bnez blez bgez bltz bgtz bgt ble bgtu bleu`
- `mv li la sext.w neg negw seqz snez sltz sgtz`

Recap III

RISC-V assembly directives

- `.asciz .string .zero`
- `.text .data .rodata .bss .comm .common .section`
- `.globl .local .equ .align .balign .byte .half .word .dword`
- `.option .file .ident .size .incbin`

Dealing with an Array

Declaration

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

Remark

- Similar to the definition of array in C++, "`a`" is the address of the first element of the array.
- Rest of the elements can be accessed through with *.word* offset (i.e., 4 bytes). (What should be the offset for the 2nd element in the array above?)

Function Call Procedure I

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.

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 II

JALR

It is similar to *jal*. The return address (i.e., the PC, which is the address of the instruction following the JALR) is saved in the destination register.

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, 0(x1)           #  assign 'PC + 4' to x1 and jump to x1 + 0
    with a appropriate alignment
```

Function Call Procedure III

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)
```

Function Call Procedure IV

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'
```

Function Call Procedure V

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

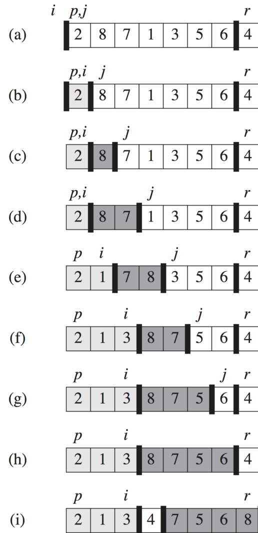
It is similar to BENE, BLT, BLTU, BGE, BGEU...

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).

```
1:  function PARTITION(A, lo, hi)
2:      pivot ← A[hi]
3:      i ← lo-1;
4:      for j = lo; j ≤ hi-1; j ← j+1 do
5:          if A[j] ≤ pivot then
6:              i ← i+1;
7:              swap A[i] with A[j];
8:          end if
9:      end for
10:     swap A[i+1] with A[hi];
11:     return i+1;
12: end function
```

Example of Partition



∗In this example, p = lo and r = hi.

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

Submission Method:

Submit the source code and report after the whole Lab1, onto blackboard.

Appendix-A Simple Sort Example

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
```