付録A

Go言語で実装したアセンブラ

main.go

```
package main

import (
        "bufio"
        "io"
        "log"
        "os"

        "github.com/kawasin73/computer-architecture-3s/asm"
        "github.com/pkg/errors"
)

func main() {
        a := asm.NewAssembler()
        if len(os.Args) > 1 {
                inputs := os.Args[1:]
                for _, input := range inputs {
                        if err := load(a, input); err != nil {
                                log.Printf("failed to load (%v) %v\n",
input, err)
                                return
                        }
                }
                bw := bufio.NewWriter(os.Stdout)
                for _, input := range inputs {
                        if err := assemble(a, bw, input); err != nil {
                                log.Printf("failed to assemble (%v) %v\n",
input, err)
                        }
                }
                if err := bw.Flush(); err != nil {
                        log.Println("failed to flush ", err)
                }
        } else {
                // TODO: from stdin
        }
}

func load(a *asm.Assembler, file string) error {
        f, err := os.Open(file)
        if err != nil {
                return errors.Wrap(err, "open input file")
        }
        defer f.Close()
```

```
        if err = a.LoadLabels(f); err != nil {
                return errors.Wrap(err, "load labels")
        }
        return nil
}

func assemble(a *asm.Assembler, w io.Writer, file string) error {
        f, err := os.Open(file)
        if err != nil {
                return errors.Wrap(err, "open input file")
        }
        defer f.Close()

        if err = a.Assemble(f, w); err != nil {
                return errors.Wrap(err, "assemble")
        }
        return nil
}
```

asm/main.go

```
package asm

import (
        "bufio"
        "fmt"
        "io"
        "strconv"
        "strings"

        "github.com/pkg/errors"
)

func split(c rune) bool {
        switch c {
        case ' ', '\t', ',':
                return true
        default:
                return false
        }
}

// Assembler build 0,1 string from assemble language.
// Have 2 steps LoadLabels and Assemble.
// Need to LoadLabels before Assemble if text contains labels
type Assembler struct {
        i      int
        li     int
        labels map[string]int
```

```
}

func NewAssembler() *Assembler {
        return &Assembler{
                labels: make(map[string]int),
        }
}

func (a *Assembler) Reset() {
        a.i = 0
        a.li = 0
        a.labels = make(map[string]int)
}

// LoadLabels load full text and build label to line number map.
func (a *Assembler) LoadLabels(r io.Reader) error {
        scanner := bufio.NewScanner(r)
        for ; scanner.Scan(); a.li++ {
                text := scanner.Text()
                text = strings.Replace(text, ":", " : ", -1)
                line := strings.FieldsFunc(text, split)
                if len(line) < 2 {
                        return fmt.Errorf("too short line : (%v)", text)
                }
                if line[1] == ":" {
                        a.labels[line[0]] = a.li
                }
        }
        return nil
}

// Assemble load full text and write 0,1 string
// comment will be follows `;`
func (a *Assembler) Assemble(r io.Reader, w io.Writer) error {
        scanner := bufio.NewScanner(r)

        for ; scanner.Scan(); a.i++ {
                text := scanner.Text()
                text = strings.Replace(text, ":", " : ", -1)
                commentIndex := strings.Index(text, ";")
                if commentIndex != -1 {
                        // remove comment
                        text = text[:commentIndex]
                }
                line := strings.FieldsFunc(text, split)
                if len(line) == 0 {
                        continue
                } else if len(line) < 2 {
                        return fmt.Errorf("too short line : (%v)", text)
                }
                if line[1] == ":" {
                        line = line[2:]
                }
```

```
		var err error
		switch line[0] {
		case "add":
			err = a.r3Render(w, line, 0, 0)
		case "addi":
			err = a.r2immRender(w, line, 1)
		case "sub":
			err = a.r3Render(w, line, 0, 2)
		case "lui":
			err = a.r1immRender(w, line, 3)
		case "and":
			err = a.r3Render(w, line, 0, 8)
		case "andi":
			err = a.r2immRender(w, line, 4)
		case "or":
			err = a.r3Render(w, line, 0, 9)
		case "ori":
			err = a.r2immRender(w, line, 5)
		case "xor":
			err = a.r3Render(w, line, 0, 10)
		case "xori":
			err = a.r2immRender(w, line, 6)
		case "nor":
			err = a.r3Render(w, line, 0, 11)
		case "sll":
			err = a.r2Render(w, line, 0, 16)
		case "srl":
			err = a.r2Render(w, line, 0, 17)
		case "sra":
			err = a.r2Render(w, line, 0, 18)
		case "lw":
			err = a.dplRender(w, line, 16)
		case "lh":
			err = a.dplRender(w, line, 18)
		case "lb":
			err = a.dplRender(w, line, 20)
		case "sw":
			err = a.dplRender(w, line, 24)
		case "sh":
			err = a.dplRender(w, line, 26)
		case "sb":
			err = a.dplRender(w, line, 28)
		case "beq":
			err = a.r2labelRender(w, line, 32)
		case "bne":
			err = a.r2labelRender(w, line, 33)
		case "blt":
			err = a.r2labelRender(w, line, 34)
		case "ble":
			err = a.r2labelRender(w, line, 35)
		case "j":
			err = a.labelRender(w, line, 40)
		case "jal":
			err = a.labelRender(w, line, 41)
```

```
                case "jr":
                        err = a.r1Render(w, line, 42, 0)
                default:
                        err = fmt.Errorf("invalid operator")
                }
                if err != nil {
                        return errors.Wrapf(err, "(line: %v) : %v", a.i+1,
line)
                }
        }
        return nil
}

// op rd rs rt  -> op(6) rs(5) rt(5) rd(5) aux(11)
func (a *Assembler) r3Render(w io.Writer, line []string, op, aux int)
error {
        if len(line) != 4 {
                return errors.Errorf("too short elements. expected %v
elements", 4)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if err = renderR(w, line[2], line[3], line[1]); err != nil
{
                return errors.Wrap(err, "render r")
        } else if err = render(w, 11, aux); err != nil {
                return errors.Wrap(err, "render aux")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}

// op rd rs shift -> op(6) rs(5) "r0"(5) rd(5) shift(5) aux(6)
func (a *Assembler) r2Render(w io.Writer, line []string, op, aux int)
error {
        if len(line) != 4 {
                return errors.Errorf("too short elements. expected %v
elements", 4)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if err = renderR(w, line[2], "r0", line[1]); err != nil {
                return errors.Wrap(err, "render r")
        } else if num, err := strconv.Atoi(line[3]); err != nil {
                return errors.Wrap(err, "parse aux")
        } else if err = render(w, 5, num); err != nil {
                return errors.Wrap(err, "render aux")
        } else if err = render(w, 6, aux); err != nil {
                return errors.Wrap(err, "render aux")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}
```

```
// op rs -> op(6) rs(5) "r0"(5) "r0"(5) aux(11)
func (a *Assembler) r1Render(w io.Writer, line []string, op, aux int)
error {
        if len(line) != 2 {
                return errors.Errorf("too short elements. expected %v
elements", 2)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if err = renderR(w, line[1], "r0", "r0"); err != nil {
                return errors.Wrap(err, "render r")
        } else if err = render(w, 11, aux); err != nil {
                return errors.Wrap(err, "render aux")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}

// op rt rs imm -> op(6) rs(5) rt(5) imm(16)
func (a *Assembler) r2immRender(w io.Writer, line []string, op int) error
{
        if len(line) != 4 {
                return errors.Errorf("too short elements. expected %v
elements", 4)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if err = renderR(w, line[2], line[1]); err != nil {
                return errors.Wrap(err, "render r")
        } else if imm, err := strconv.Atoi(line[3]); err != nil {
                return errors.Wrap(err, "parse imm")
        } else if err = render(w, 16, imm); err != nil {
                return errors.Wrap(err, "render imm")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}

// op rt imm -> op(6) "r0"(5) rt(5) imm(16)
func (a *Assembler) r1immRender(w io.Writer, line []string, op int) error
{
        if len(line) != 3 {
                return errors.Errorf("too short elements. expected %v
elements", 3)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if err = renderR(w, "r0", line[1]); err != nil {
                return errors.Wrap(err, "render r")
        } else if imm, err := strconv.Atoi(line[2]); err != nil {
                return errors.Wrap(err, "parse imm")
        } else if err = render(w, 16, imm); err != nil {
                return errors.Wrap(err, "render imm")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
```

```
        }
        return nil
}

// op rt dpl(rs) -> op(6) rs(5) rt(5) dpl(16)
func (a *Assembler) dplRender(w io.Writer, line []string, op int) error {
        if len(line) != 3 {
                return errors.Errorf("too short elements. expected %v
elements", 3)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if base, dpl, err := parseDpl(line[2]); err != nil {
                return errors.Wrap(err, "parse dpl")
        } else if err = renderR(w, "r"+strconv.Itoa(base), line[1]); err
!= nil {
                return errors.Wrap(err, "render r")
        } else if err = render(w, 16, dpl); err != nil {
                return errors.Wrap(err, "render imm")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}

// op rt rs label -> op(6) rs(5) rt(5) {label - i -1}(16)
func (a *Assembler) r2labelRender(w io.Writer, line []string, op int)
error {
        if len(line) != 4 {
                return errors.Errorf("too short elements. expected %v
elements", 4)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if err = renderR(w, line[2], line[1]); err != nil {
                return errors.Wrap(err, "render r")
        } else if label, ok := a.labels[line[3]]; !ok {
                return fmt.Errorf("not found label %v", line[3])
        } else if err = render(w, 16, label-a.i-1); err != nil {
                return errors.Wrap(err, "render label dpl")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}

// op label -> op(6) label(26)
func (a *Assembler) labelRender(w io.Writer, line []string, op int) error
{
        if len(line) != 2 {
                return errors.Errorf("too short elements. expected %v
elements", 2)
        } else if err := render(w, 6, op); err != nil {
                return errors.Wrap(err, "render op")
        } else if label, ok := a.labels[line[1]]; !ok {
                return fmt.Errorf("not found label %v", line[1])
```

```
        } else if err = render(w, 26, label); err != nil {
                return errors.Wrap(err, "render label dpl")
        } else if err = renderEnd(w); err != nil {
                return errors.Wrapf(err, "render end")
        }
        return nil
}

func parseDpl(v string) (base, dpl int, err error) {
        r := strings.NewReader(v)
        _, err = fmt.Fscanf(r, "%d(r%d)", &dpl, &base)
        return
}

func render(w io.Writer, digit, num int) error {
        unum := uint(num)
        format := "%0" + strconv.Itoa(digit) + "b_"
        bin := fmt.Sprintf(format, unum)
        _, err := fmt.Fprint(w, bin[len(bin)-digit-1:])
        return err
}

func renderR(w io.Writer, rs ...string) error {
        for _, r := range rs {
                if r[0] != 'r' {
                        return fmt.Errorf("invalid register(%v)", r)
                } else if num, err := strconv.Atoi(r[1:]); err != nil {
                        return errors.Wrapf(err, "convert to num
register(%v)", r)
                } else if err = render(w, 5, num); err != nil {
                        return errors.Wrap(err, "write register")
                }
        }
        return nil
}

func renderEnd(w io.Writer) error {
        _, err := fmt.Fprintln(w)
        return err
}
```