CA – project

Make LC-2K Assembler , Simulator

What is LC-2K?

- Very simple but enough to solve problem.
- Has a 8 register
- 32-bit computer
- 65536 word-memory
- All addresses are Word-addresses
- NOTE : reg 0 always contain 0.

LC-2K ISA

- R-Type
-

R-type instructions (add, nor):
bits 24-22: opcode
bits 21-19: reg A
bits 18-16: reg B
bits 15-3: unused (should all be 0)
bits 2-0: destReg

LC-2K ISA

add (R-type format)	000	Add contents of regA with contents of regB, store results in destReg.
nor (R-type format)	001	Nor contents of regA with contents of regB, store results in destReg. This is a bitwise nor; each bit is treated independently.

LC-2K ISA

- I-Type

I-type instructions (lw, sw, beq):
bits 24-22: opcode
bits 21-19: reg A
bits 18-16: reg B
bits 15-0: offsetField (a 16-bit, 2's complement number with a range of -32768 to 32767)

LC-2K ISA

lw (I-type format) 010	Load regB from memory. Memory address is formed by adding offset-Field with the contents of regA.
sw (I-type format) 011	Store regB into memory. Memory address is formed by adding offset-Filed with the contents of regA.
beq (I-type format) 100	If the contents of regA and regB are the same, then branch to the address PC+1+offset-Field, where PC is the address of this beq instruction.

LC-2K ISA

- J-Type

J-type instructions (jalr):
bits 24-22: opcode
bits 21-19: reg A
bits 18-16: reg B
bits 15-0: unused (should all be 0)

LC-2K ISA

jalr (J-type format) 101	First store PC+1 into regB, where PC is the address of this jalr instruction. Then branch to the address contained in regA. Note that this implies if regA and regB refer to the same register, the net effect will be jumping to PC+1.

LC-2K ISA

- O-Type
-

O-type instructions (halt, noop):
bits 24-22: opcode
bits 21-0: unused (should all be 0)

LC-2K ISA

halt (O-type format) 110	Increment the PC (as with all instructions), then halt the machine (let the simulator notice that the machine haled).
noop (O-type format) 111	Do nothing.

LC-2K

- Make Assembler (40%)
- Make Simulator (40%)
- Multiplication (20%)

1. Make Assembler

- Write program to take an assembly-language a nd translate it into machine code.
- And you need to translate symbolic name for ad dress into numeric Values

Assembly Format

- Label<white>instruction<withe>field0<white>field1<whit e>field2<white>comments

```
start    add 1 2 1          decrement reg1
```

Assembly Format

- Label is Optional (the white space following the label field is required)
- Label’s name has a maximum 6 char and can c onsist of letters and number but first character must be letter
- Unused field should be skip.

Symbolic Address

```
        lw 0 1 five     load reg1 with 5(symbolic addr)
        lw 1 2 3        load reg2 with -1(numeric addr)
start   add 1 2 1       decrement reg1
        beq 0 1 2       goto end of program when reg1==0
        beq 0 0 start   start go back to the beginning of the loop
        noop
done    halt            end of prog
five    .fill 5
neg1    .fill -1
stAddr  .fill start
```

Symbolic address == label , so assembler should compute offset-Field to be equal to the address of the label

.fill →tells the assembler to put a number into the place where the instruction would normally be stored.

Make Assembler

```
        lw 0 1 five     load reg1 with 5(symbolic addr)
        lw 1 2 3        load reg2 with -1(numeric addr)
start   add 1 2 1       decrement reg1
        beq 0 1 2       goto end of program when reg1==0
        beq 0 0         start go back to the beginning of the loop
        noop
done    halt            end of prog
five    .fill 5
neg1    .fill -1
stAddr  .fill start
```


Assembler

Machine Code.

```
        lw 0 1 five     load reg1 with 5(symbolic addr)
        lw 1 2 3        load reg2 with -1(numeric addr)
start   add 1 2 1       decrement reg1
        beq 0 1 2       goto end of program when reg1==0
        beq 0 0         start go back to the beginning of the loop
        noop
done    halt            end of prog
five    .fill 5
neg1    .fill -1
stAddr  .fill start
```

test.as

My_Assembler test.as test.mc

And here is the corresponding machine language:

(address 0): 8454151 (hex 0x810007)
(address 1): 9043971 (hex 0x8a0003)
(address 2): 655361 (hex 0xa0001)
(address 3): 16842754 (hex 0x1010002)
(address 4): 16842749 (hex 0x100fffd)
(address 5): 29360128 (hex 0x1c00000)
(address 6): 25165824 (hex 0x1800000)
(address 7): 5 (hex 0x5)
(address 8): -1 (hex 0xffffffff)
(address 9): 2 (hex 0x2)

test.mc

```
8454151
9043971
655361
16842754
16842749
29360128
25165824
5
-1
2
```

Make Assembler

lw 0 1 five →00..00 010 000 001 0000000000000111 → 8454151

Unuse op regA regB offset-field

lw 1 2 3 →00..00 010 001 010 0000000000000011 → 9043971

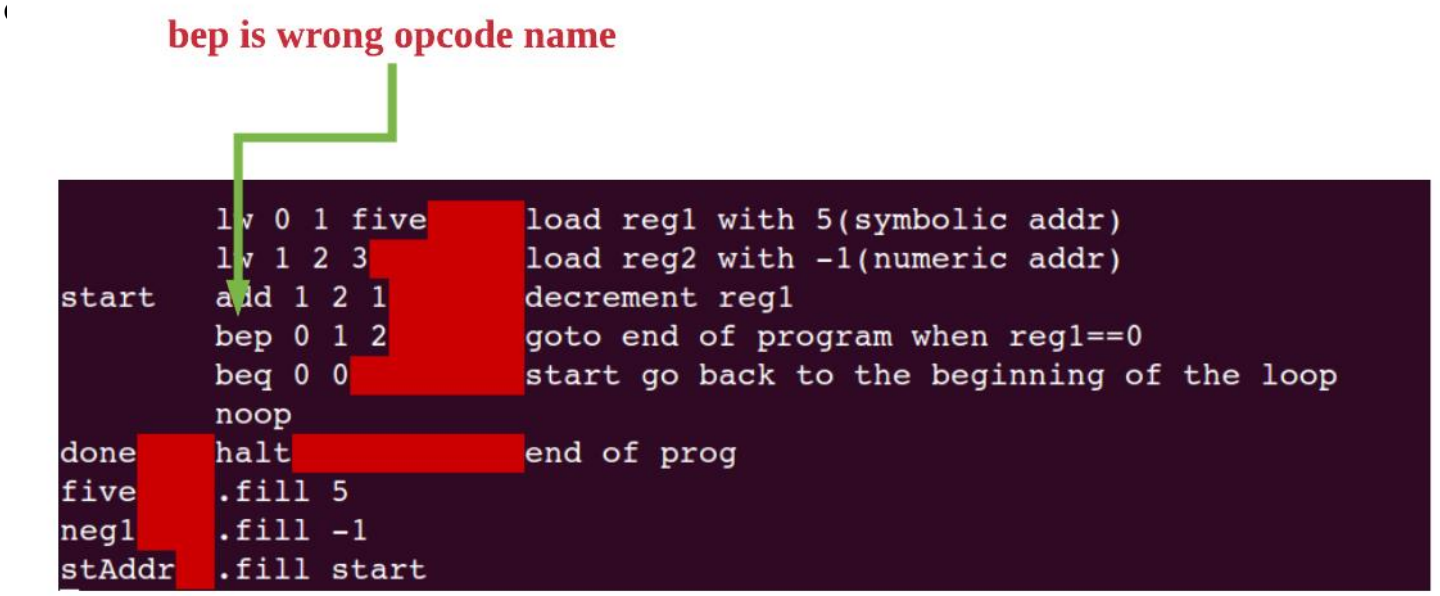
Unuse op regA regB offset-field

Add 1 2 1→00..00 000 001 010 0000000000000 01 → 655361

Unuse op regA regB Unuse dest-reg

Make Assembler

- NOTE : implement Error Check




```
choi@choi-com:~/Desktop/CA/컴구 Project/Project_1$ ./assembler as.as mc
error: unrecognized opcode
bep
```

Make Simulator

And here is the corresponding machine language:

(address 0): 8454151 (hex 0x810007)
(address 1): 9043971 (hex 0x8a0003)
(address 2): 655361 (hex 0xa0001)
(address 3): 16842754 (hex 0x1010002)
(address 4): 16842749 (hex 0x100fffd)
(address 5): 29360128 (hex 0x1c00000)
(address 6): 25165824 (hex 0x1800000)
(address 7): 5 (hex 0x5)
(address 8): -1 (hex 0xffffffff)
(address 9): 2 (hex 0x2)

./Sim test.mc > Result

Each program counter(PC) , display memory, reg state , and PC

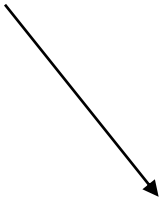
When initializing simulator , register and PC should be Zero.

And stop when meet halt Inst.

Make Simulator

```
memory[0]=8454151
memory[1]=9043971
memory[2]=655361
memory[3]=16842754
memory[4]=16842749
memory[5]=29360128
memory[6]=25165824
memory[7]=5
memory[8]=-1
memory[9]=2
```

test.mc



./sim test.mc > Result

Result

```
@@@
state:
        pc 0
        memory:
                mem[ 0 ] 8454151
                mem[ 1 ] 9043971
                mem[ 2 ] 655361
                mem[ 3 ] 16842754
                mem[ 4 ] 16842749
                mem[ 5 ] 29360128
                mem[ 6 ] 25165824
                mem[ 7 ] 5
                mem[ 8 ] -1
                mem[ 9 ] 2
        registers:
                reg[ 0 ] 0
                reg[ 1 ] 0
                reg[ 2 ] 0
                reg[ 3 ] 0
                reg[ 4 ] 0
                reg[ 5 ] 0
                reg[ 6 ] 0
                reg[ 7 ] 0
end state
```

Make Simulator

```
@@@
state:
        pc 4
        memory:
                mem[ 0 ] 8454151
                mem[ 1 ] 9043971
                mem[ 2 ] 655361
                mem[ 3 ] 16842754
                mem[ 4 ] 16842749
                mem[ 5 ] 29360128
                mem[ 6 ] 25165824
                mem[ 7 ] 5
                mem[ 8 ] -1
                mem[ 9 ] 2
        registers:
                reg[ 0 ] 0
                reg[ 1 ] 4
                reg[ 2 ] -1
                reg[ 3 ] 0
                reg[ 4 ] 0
                reg[ 5 ] 0
                reg[ 6 ] 0
                reg[ 7 ] 0
end state
```

```
@@@
state:
        pc 7
        memory:
                mem[ 0 ] 8454151
                mem[ 1 ] 9043971
                mem[ 2 ] 655361
                mem[ 3 ] 16842754
                mem[ 4 ] 16842749
                mem[ 5 ] 29360128
                mem[ 6 ] 25165824
                mem[ 7 ] 5
                mem[ 8 ] -1
                mem[ 9 ] 2
        registers:
                reg[ 0 ] 0
                reg[ 1 ] 0
                reg[ 2 ] -1
                reg[ 3 ] 0
                reg[ 4 ] 0
                reg[ 5 ] 0
                reg[ 6 ] 0
                reg[ 7 ] 0
end state
```

Make Multiplication

- Write Assembly language program to multiply t wo number (32766 * 10838).
- Input Number by reading memory loaction (Lab el) called “mcand” and “mplier”.
- The result should be stored in reg1 when progra m halt
- Note : two input number are at most 15bits and positive number.
- For efficient , it must be line <= 50 and Inst <= 1 000

More info.

- designer9406@gmail.com – 엄홍준