Project 1 Sketch

2018년 10월 17일 수요일　　오후 7:42

```
        .data
data1:  .word   100          0x1000 0000
data2:  .word   200          0x1000 0004
data3:  .word   0x12345678
        .text
main:
  00    and     $17, $17, $0
  04    and     $18, $18, $0
        la      $8, data1  ]     09  lui $8, 4096
        la      $9, data2  ] =>  0C  lui $9, 4096
                                 10  ori $9, $9, 4
  14    and     $10, $10, $0
lab1:
  18    and     $11, $11, $0
lab2:
  1C    addiu   $17, $17, 0x1      -3    1
  20    addiu   $11, $11, 0x1            2
  24    or      $9, $9, $0               3
  28    bne     $11, $8, lab2            4
lab3:
  2C    addiu   $18, $18, 0x2
  30    addiu   $11, $11, 1        -5
  34    sll     $18, $17, 1
  38    srl     $17, $18, 1
  3C    and     $19, $17, $18
  40    bne     $11, $9, lab3
lab4:
  44    addu    $5, $5, $31
  48    nor     $16, $17, $18
  4C    beq     $10, $8, lab5
  50    j       lab1               +1
lab5:
  54    ori     $16, $16, 0xf0f0
```

data1의 메모리 주소 0x1000 0000의 상위 16비트를 $8에 넣는다.

1100 1000 0000 0100

① data section에서 data를 읽고 map으로 저장. (KVP)

② ~~data 개수 * 4로 data section size 구하기.~~

③ 파일 전체를 스캔해 각 라벨의 labels 구조체에 담는다.

④ text 첫 라인부터 순서대로 read하며 각 instruction과 arguments를 text 구조체에 담는다.

⑤ text 구조체를 순회하며 instruction을 보고 그에 맞는 포맷으로 binary digits를 만든다.

→ op가 0이라면 R-format
2 or 3이라면 J-format
그 외에는 I-format

Initialization

① instruction을 구성하는 알파벳의 유니코드 자리에 opcode를 저장.

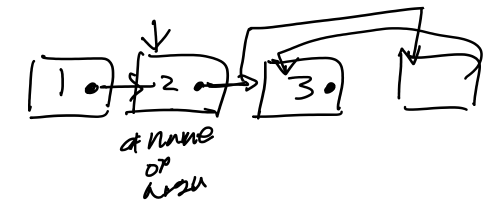
[0, 0, 0, 0, 0, 0, 0, 0, 0, …

ex) add → a 97 − 96 = 1
d 100 − 96 = 4
d 100 − 96 = 4
(1 + 4 + 4)

initialize 함수에서 instruction table 초기화. 각 instruction의 ASCII를 기반으로 hash table 구현.

중복시, →

② instruction line을 읽어줄때는 첫 명령의 instruction table index를 참고, 해당 instruction의 op, funct, arguments를 넣어준다.



instruction table structure

```
typedef struct {
    int pointer
    int opcode
    int funct
} INSTRUCTION_TABLE

index = inst[0] - 96 + inst[end-1] - 96
init_index = index
for ( k = index; insts[k].pointer != -1; k++) {   // insts[k]에 자리가 없을 때까지
    index += 1   // 다음 자리로 커서 이동
}
insts[init_index].pointer = index   // 초기 index 포인터에 다음자리 위치 저장
insts[index].opcode ~~
insts[index].funct ~~
insts[index].pointer = -1   // 현재 자리에 값이 있음을 표현
```

이후 instruction을 resolve할 때 pointer를 확인하면 O(1) 시간에 해당 위치로 이동할 수 있다

output:

text section size = 88　　data section size = 12

and $17 $0 $17 0 24HEX 36DEC
op rs rt rd shamt funct

```
00000000000000000000000001011000 00000000000000000000000000001100 000000 10001 00000 10001 00000 100100
and $18 $0 $18 / lui (0xF) $8 4096 / lui (0xF) $9 4096
000000 10010 00000 10010 00000 100100 001111 00000 01000 0001000000000000 001111 00000 01001 0001000000000000
ori (0xD) $9 $9 4 / and $10 $0 $10 0x24 / and $11 $0 $11 0x24
001101 01001 01001 0000000000000100 000000 01010 00000 01010 00000 100100 000000 01011 00000 01011 00000 100100
addiu (0x9) $17 $17 0x1
001001 10001 10001 0000000000000001 00100101011010110000000000000001 00000010010000000100100000100101
bne (0x5) $11 $8 -3 / addiu (0x9)
000101 01011 01000 1111111111111100 001001 10010100100000000000000010 00100101011010110000000000000001
sll $17 $18 1
000000 00000 10001 10010 00001 000000 00000000000100101000100001000010 00000010001100101001100000100100
bne (0x5) -5 / 17 18 16
000101 01011 01001 1111111111111010 00000000101111110010100000100001 000000 10001 10010 10000 00000 100111
beq +1 / j (0x2) 0x100006 << 2 = 0x400018 / ori (0xD)
000100 01010 01000 0000000000000001 000010 00000100000000000000000110 001101 10000100001111000011110000
100 / 200 / 0x12345678
00000000000000000000000001100100 00000000000000000000000011001000 00010010001101000101011001111000
```

TODO

~~① la를 lui, ori로 분리~~ → ~~어떻게?~~

~~② 파일 앞에 section size 붙이기~~

~~③ 파일 끝에 data 붙이기~~

④ 배열 표현 처리하기 4($8) → disassemble instructions에 postfix가 ')'인 경우 추가.

```
        .data
data1:  .word   0x12c
data2:  .word   0xc8
        .text
main:
  00    and     $10, $10, $0
  04    and     $11, $11, $0
        la      $8, data1
        la      $9, data2
        addiu   $10, $10, 0x1
        sll     $10, $10, 1
        sll     $11, $11, 1
loop:
        addiu   $10, $10, 0x1
        addiu   $11, $11, 1
        or      $9, $9, $0
        subu    $18, $18, $10
        sll     $18, $17, 1
        sll     $17, $18, 1
        addu    $11, $11, $31
        nor     $16, $17, $18
        bne     $11, $8, loop
        j       exit          000010   000001000000000000000010011
exit:
        andi    $15, $15, 0x0f
```