```
# Stephen Ermshar
# CPTR 280
# Final Project : get inst 2
# 2017 NOV 24
# version history: https://github.com/sermshar/cptr280

.data
    # https://opencores.org/project,plasma,opcodes
    # used a spreadsheet and textwrangler to turn the table in the link into these two lines
    # NOTE: many instructions share a common initial 6 bits, like sycall and sll (000000) and differ in the last bits.

    # TODO: add special case for nop, where the entire instruction is 0x0, check for nop before checking for dup_00 becuase sll has 0x00 as its first and last 6 bits, a nop would be interpreted as an sll

    dup_00_fnc:         .byte       0x1A, 0x1B, 0x11, 0x13, 0x18, 0x19, 0x08, 0x0C, 0x0D, 0x20, 0x21, 0x24, 0x27, 0x25, 0x2A, 0x2B, 0x22, 0x23, 0x26, 0x00, 0x04, 0x03, 0x07, 0x02, 0x06, 0x10, 0x12, 0x09
    dup_00_fnc_txt:     .asciiz     "div    ", "divu   ", "mthi   ", "mtlo   ", "mult   ", "multu  ", "jr     ", "syscall", "break  ", "add    ", "addu   ", "and    ", "nor    ", "or     ", "slt    ", "sltu   ", "sub    ", "subu   ", "xor    ", "sll    ", "sllv   ", "sra    ", "srav   ", "srl    ", "srlv   ", "mfhi   ", "mflo   ", "jalr   "
    dup_00_cases:       .byte       2, 2, 3, 3, 2, 2, 3, 10, 10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 4, 4, 3

    dup_01:             .byte       0x01, 0x11, 0x00, 0x10
    dup_01_txt:         .asciiz     "bgez   ", "bgezal ", "bltz   ", "bltzal "
    dup_01_cases:       .byte       6, 6, 6, 6

    dup_10:             .byte       0x00, 0x04
    dup_10_txt:         .asciiz     "mfc0   ", "mtc0   "
    dup_10_cases:       .byte       7, 7

    not_dup_opc:        .byte       0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F, 0x20, 0x21, 0x23, 0x24, 0x25, 0x28, 0x29, 0x2B
    not_dup_opc_txt:    .asciiz     "j      ", "jal    ", "beq    ", "bne    ", "blez   ", "bgtz   ", "addi   ", "addiu  ", "slti   ", "sltiu  ", "andi   ", "ori    ", "xori   ", "lui    ", "lb     ", "lh     ", "lw     ", "lbu    ", "lbu    ", "sb     ", "sh     ", "sw     "
    not_dup_cases:      .byte       8, 8, 9, 9, 6, 6, 9, 9, 9, 9, 9, 9, 9, 11, 12, 12, 12, 12, 12, 12, 12, 12

    nl:         .asciiz     "\n"
    tb:         .asciiz     "\t"
    the_nop:    .asciiz     "nop    "
    dollar:     .asciiz     "$"
    comma:      .asciiz     ","
    l_paren:    .asciiz     "("
    r_paren:    .asciiz     ")"
.text
    main:
        la      $s0,    0x00400000      # start with the first instruction

### MAIN LOOP
    main_loop:
        add     $s7,    $0,     $0          # set s7 to 0 in case it got messed with

        jal     get_inst                    # get the instruction and PUT IT IN $s1
        nop

        jal     check_if_last               # check if it's the last instruction, print it and quit if it is
```

```
49          nop
50
51          jal     print_hex_inst              # print the instruction (hex integer)
52          nop
53
54          jal     check_if_nop                # checks for the special case of nop, which
 … confuses the system for sll
55          nop
56
57          jal     print_opc
58          nop
59
60      was_nop:                                # kinda like TSA Precheck, just for nop, because
 … it's special like that
61          la      $a0,    nl
62          li      $v0,    4               # syscall_4: print string newline
63          syscall
64
65          bne     $s7,    $0,     end     # a 1 in s7 indicates that this was the last loop
 … DO NOT USE s7 WITHOUT PUTTING IT BACK!
66
67          addi    $s0,    $s0,    4       # go to next instruction
68          j main_loop                     # loop again
69          nop
70  ### ------
71
72  ### PRINT OPC :
73      print_opc:      # THIS DOESN'T DO THE PRINTING, IT JUST STARTS THE PROCESS OF
 … PRINTING THE WHOLE INSTRUCTION, NEED TO CHANGE NAME, BUT NOT RIGHT NOW
74          addi    $sp,    $sp,    -4
75          sw      $ra,    0($sp)
76
77          li      $a0,    0
78          li      $a1,    5   # 0 - 5 are first 6 bits of the register
79          jal     splice_bits # splice result stored in $t0
80          nop
81
82          la      $ra,    print_opc_out       # instead of using a jal I'm setting $ra
 … myself becuase I dont want any of these to return where they left off, but I also want to
 … convert everything to proper jumps with links so I can use $sp more consistently
83
84          beq     $t0,    $0,     dup_00_case     # if opcode is 0x00, then $t0 will be all
 … 0, no need to create an immediate register
85          nop
86
87          li      $t1,    0x01                # immediate register to see if opcode is 0x01
88          beq     $t0,    $t1,    dup_01_case
89          nop
90
91          li      $t1,    0x10
92          beq     $t0,    $t1,    dup_10_case
93          nop
94
95          j not_dup_case                          # if the first 6 bits don't fit in the
 … first three conditionals, they must not be from a dup set
96          nop
97
98      print_opc_out:
99          lw      $ra,    0($sp)
100         addi    $sp,    $sp,    4
101         jr      $ra
102         nop
103 ### ------
104
```

```
105 ### DUP 00 : handles cases where the opcode is 0x00
106     dup_00_case:
107         addi    $sp,    $sp,    -4
108         sw      $ra,    0($sp)
109 
110         # resplice to get the function code (last 6 bits)
111         li      $a0,    26
112         li      $a1,    31  # 26 - 31 are last 6 bits of the register
113         jal     splice_bits # splice result stored in $t0
114         nop
115 
116         la      $t1,    dup_00_fnc
117         la      $t2,    dup_00_fnc_txt
118         # make search JAL after adding jr and stack stuff to each case
119         jal     search
120         nop
121         li      $v0,    4               # syscall_4: print string
122         syscall
123 
124         la      $t7,    dup_00_cases    # base address for the register printing cases
125         jal print_regs
126         nop
127 
128         lw      $ra,    0($sp)
129         addi    $sp,    $sp,    4
130         jr      $ra
131         nop
132 ### ------
133 
134 ### DUP 01 : handles cases where the opcode is 0x01
135     dup_01_case:
136         addi    $sp,    $sp,    -4
137         sw      $ra,    0($sp)
138 
139         # resplice to get the 5 bits in 3rd field
140         li      $a0,    11
141         li      $a1,    15  # 11 - 15 are 5 bits of the register
142         jal     splice_bits # splice result stored in $t0
143         nop
144 
145         la      $t1,    dup_01
146         la      $t2,    dup_01_txt
147         jal     search
148         nop
149         li      $v0,    4               # syscall_4: print string
150         syscall
151 
152         la      $t7,    dup_01_cases    # base address for the register printing cases
153         jal print_regs
154         nop
155 
156         lw      $ra,    0($sp)
157         addi    $sp,    $sp,    4
158         jr      $ra
159         nop
160 
161 ### ------
162 
163 ### DUP 10 : handles cases where the opcode is 0x10
164     dup_10_case:
165         addi    $sp,    $sp,    -4
166         sw      $ra,    0($sp)
167 
168         # resplice to get the 5 bits in 2nd field
```

```
169         li      $a0,    6
170         li      $a1,    10  # 6 - 10 are 5 bits of the register
171         jal     splice_bits # splice result stored in $t0
172         nop
173
174         la      $t1,    dup_10
175         la      $t2,    dup_10_txt
176         jal     search
177         nop
178         li      $v0,    4               # syscall_4: print string
179         syscall
180
181         la      $t7,    dup_10_cases    # base address for the register printing cases
182         jal print_regs
183         nop
184
185         lw      $ra,    0($sp)
186         addi    $sp,    $sp,    4
187         jr      $ra
188         nop
189 ### ------
190
191 ### NOT DUP : handles cases where the opcode is unique
192     not_dup_case:
193         addi    $sp,    $sp,    -4
194         sw      $ra,    0($sp)
195         # resplice to get first 6 bits in the opcode
196         # this is a repeat of a splice that was done earlier, but to reduce errors I'm
  … sticking with this less efficient route. once i know it works I may refactor it so as to
  … save the opcode splice instead of resplicing to get the unique opcode again
197         li      $a0,    0
198         li      $a1,    5   # 0 - 5 are first 6 bits of the register
199         jal     splice_bits # splice result stored in $t0
200         nop
201
202         la      $t1,    not_dup_opc
203         la      $t2,    not_dup_opc_txt
204         jal     search
205         nop
206         li      $v0,    4               # syscall_4: print string
207         syscall
208
209         la      $t7,    not_dup_cases   # base address for the register printing cases
210         jal print_regs
211         nop
212
213         lw      $ra,    0($sp)
214         addi    $sp,    $sp,    4
215         jr      $ra
216         nop
217 ### ------
218
219 ### PRINT REGS : print the rest of the instruction
220     print_regs:
221         addi    $sp,    $sp,    -4
222         sw      $ra,    0($sp)
223
224         # we need the base address of the relevant case list (t7)
225         # we need the current offset we used to find the opcode txt (t6)
226         # use t9 as an immediate register
227
228         add     $t7,    $t7,    $t6 # add the address and the offset to get the address
  … of the case number
229         lb      $t6,    0($t7)      # load the case number into t6, it should be safe to
```

```
clober t6 now that we've gotten the address from it, we wont be using the offset again
for the current instruction

        li      $t9,    0
        beq     $t6,    $t9,    c_0
        nop

        li      $t9,    1
        beq     $t6,    $t9,    c_1
        nop

        li      $t9,    2
        beq     $t6,    $t9,    c_2
        nop

        li      $t9,    3
        beq     $t6,    $t9,    c_3
        nop

        li      $t9,    4
        beq     $t6,    $t9,    c_4
        nop

        li      $t9,    5
        beq     $t6,    $t9,    c_5
        nop

        li      $t9,    6
        beq     $t6,    $t9,    c_6
        nop

        li      $t9,    7
        beq     $t6,    $t9,    c_7
        nop

        li      $t9,    8
        beq     $t6,    $t9,    c_8
        nop

        li      $t9,    9
        beq     $t6,    $t9,    c_9
        nop

        li      $t9,    10
        beq     $t6,    $t9,    c_10
        nop

        li      $t9,    11
        beq     $t6,    $t9,    c_11
        nop

        li      $t9,    12
        beq     $t6,    $t9,    c_12
        nop

    print_regs_out:
        lw      $ra,    0($sp)
        addi    $sp,    $sp,    4
        jr      $ra
        nop
### ------

### D REG : print the register in the third field of the instruction
    d_reg:
```

```
292         addi    $sp,    $sp,    -8
293         sw      $ra,    0($sp)
294         sw      $t0,    4($sp)
295         # splice (third field ie. 16-20)
296         li      $a0,    16
297         li      $a1,    20
298         jal     splice_bits # splice result stored in $t0
299         nop
300
301         la      $a0,    dollar
302         li      $v0,    4               # syscall_4: print string
303         syscall
304
305         move    $a0,    $t0
306         li      $v0,    1               # syscall_1: print int
307         syscall
308
309         lw      $t0,    -4($sp)
310         lw      $ra,    0($sp)
311         addi    $sp,    $sp,    8
312         jr      $ra
313         nop
314 ### ------
315
316 ### S REG : print the register in the first field of the instruction
317     s_reg:
318         addi    $sp,    $sp,    -8
319         sw      $ra,    0($sp)
320         sw      $t0,    4($sp)
321         # splice (first field ie. 6-10)
322         li      $a0,    6
323         li      $a1,    10
324         jal     splice_bits # splice result stored in $t0
325         nop
326
327         la      $a0,    dollar
328         li      $v0,    4               # syscall_4: print string
329         syscall
330
331         move    $a0,    $t0
332         li      $v0,    1               # syscall_1: print int
333         syscall
334
335         lw      $t0,    -4($sp)
336         lw      $ra,    0($sp)
337         addi    $sp,    $sp,    8
338         jr      $ra
339         nop
340 ### ------
341
342 ### T REG : print the register in the second field of the instruction
343     t_reg:
344         addi    $sp,    $sp,    -8
345         sw      $ra,    0($sp)
346         sw      $t0,    4($sp)
347         # splice (third register to be printed is the second field ie. 11-15)
348         li      $a0,    11
349         li      $a1,    15
350         jal     splice_bits # splice result stored in $t0
351         nop
352
353         la      $a0,    dollar
354         li      $v0,    4               # syscall_4: print string
355         syscall
```

```
356
357         move    $a0,    $t0
358         li      $v0,    1               # syscall_1: print int
359         syscall
360
361         lw      $t0,    -4($sp)
362         lw      $ra,    0($sp)
363         addi    $sp,    $sp,    8
364         jr      $ra
365         nop
366 ### ------
367
368 ### SHIFT FIELD : print the integer in the fourth field of the instruction
369     shift_field:
370         addi    $sp,    $sp,    -8
371         sw      $ra,    0($sp)
372         sw      $t0,    4($sp)
373         # splice (fourth field ie. 21-25)
374         li      $a0,    21
375         li      $a1,    25
376         jal     splice_bits # splice result stored in $t0
377         nop
378
379         move    $a0,    $t0
380         li      $v0,    1               # syscall_1: print int
381         syscall
382
383         lw      $t0,    -4($sp)
384         lw      $ra,    0($sp)
385         addi    $sp,    $sp,    8
386         jr      $ra
387         nop
388 ### ------
389
390 ### IMM FIELD : print the integer in the immediate field of the instruction
391     imm_field:
392         addi    $sp,    $sp,    -8
393         sw      $ra,    0($sp)
394         sw      $t0,    4($sp)
395         # splice (bits 16-31)
396         li      $a0,    16
397         li      $a1,    31
398         jal     splice_bits # splice result stored in $t0
399         nop
400
401         move    $a0,    $t0
402         li      $v0,    1               # syscall_1: print int
403         syscall
404
405         lw      $t0,    -4($sp)
406         lw      $ra,    0($sp)
407         addi    $sp,    $sp,    8
408         jr      $ra
409         nop
410 ### ------
411
412 ### TARGET FIELD : print the integer in the target field of the instruction
413     target_field:
414         addi    $sp,    $sp,    -8
415         sw      $ra,    0($sp)
416         sw      $t0,    4($sp)
417         # splice (bits 6-31)
418         li      $a0,    6
419         li      $a1,    31
```

```
420         jal     splice_bits # splice result stored in $t0
421         nop
422
423         move    $a0,    $t0
424         li      $v0,    34              # syscall_34: print hex
425         syscall
426
427         lw      $t0,    -4($sp)
428         lw      $ra,    0($sp)
429         addi    $sp,    $sp,    8
430         jr      $ra
431         nop
432 ### ------
433
434 ##########################
435
436 # THE CASE SYSTEM :
437     # Case Numbers:
438     # 0: rd, rs, rt
439     # 1: rt, rd, sa
440     # 2: rs, rt
441     # 3: rs
442     # 4: rd
443     # 5: rs, rd
444     # 6: rs, imm
445     # 7: rt, rd
446     # 8: target
447     # 9: rt, rs, imm
448     # 10: No fields
449     # 11: rt, imm
450     # 12: rt,imm(rs)
451
452 ### 0 : rd, rs, rt
453     c_0:
454         la      $a0,    tb
455         li      $v0,    4               # syscall_4: print string
456         syscall
457
458         jal d_reg
459         nop
460
461         la      $a0,    comma
462         li      $v0,    4               # syscall_4: print string
463         syscall
464
465         la      $a0,    tb
466         li      $v0,    4               # syscall_4: print string
467         syscall
468
469         jal s_reg
470         nop
471
472         la      $a0,    comma
473         li      $v0,    4               # syscall_4: print string
474         syscall
475
476         la      $a0,    tb
477         li      $v0,    4               # syscall_4: print string
478         syscall
479
480         jal t_reg
481         nop
482         j       print_regs_out
483         nop
```

```
484 ### ------
485
486 ### 1: rd, rt, sa
487     c_1:
488         la      $a0,    tb
489         li      $v0,    4               # syscall_4: print string
490         syscall
491
492         jal d_reg
493         nop
494
495         la      $a0,    comma
496         li      $v0,    4               # syscall_4: print string
497         syscall
498
499         la      $a0,    tb
500         li      $v0,    4               # syscall_4: print string
501         syscall
502
503         jal t_reg
504         nop
505
506         la      $a0,    comma
507         li      $v0,    4               # syscall_4: print string
508         syscall
509
510         la      $a0,    tb
511         li      $v0,    4               # syscall_4: print string
512         syscall
513
514         jal shift_field
515         nop
516         j       print_regs_out
517         nop
518 ### ------
519
520 ### 2: rs, rt
521     c_2:
522         la      $a0,    tb
523         li      $v0,    4               # syscall_4: print string
524         syscall
525
526         jal s_reg
527         nop
528
529         la      $a0,    comma
530         li      $v0,    4               # syscall_4: print string
531         syscall
532
533         la      $a0,    tb
534         li      $v0,    4               # syscall_4: print string
535         syscall
536
537         jal t_reg
538         nop
539         j       print_regs_out
540         nop
541 ### ------
542
543 ### 3: rs
544     c_3:
545         la      $a0,    tb
546         li      $v0,    4               # syscall_4: print string
547         syscall
```

```
548
549         jal s_reg
550         nop
551         j       print_regs_out
552         nop
553 ### ------
554
555 ### 4: rd
556     c_4:
557         la      $a0,    tb
558         li      $v0,    4               # syscall_4: print string
559         syscall
560
561         jal d_reg
562         nop
563         j       print_regs_out
564         nop
565 ### ------
566
567 ### 5: rs, rd
568     c_5:
569
570     # I don't know where I got the idea that this case exists, I can't find it in my
  … spreadsheet now... I'll leave this here in case I remember, I'm pretty sure I double
  … checked the others so this shouldn't be indicative of a larger problem
571
572 ### ------
573
574 ### 6: rs, imm
575     c_6:
576         la      $a0,    tb
577         li      $v0,    4               # syscall_4: print string
578         syscall
579
580         jal s_reg
581         nop
582
583         la      $a0,    comma
584         li      $v0,    4               # syscall_4: print string
585         syscall
586
587         la      $a0,    tb
588         li      $v0,    4               # syscall_4: print string
589         syscall
590
591         jal imm_field
592         nop
593         j       print_regs_out
594         nop
595 ### ------
596
597 ### 7: rt, rd
598     c_7:
599         la      $a0,    tb
600         li      $v0,    4               # syscall_4: print string
601         syscall
602
603         jal t_reg
604         nop
605
606         la      $a0,    comma
607         li      $v0,    4               # syscall_4: print string
608         syscall
609
```

```
610         la      $a0,    tb
611         li      $v0,    4               # syscall_4: print string
612         syscall
613
614         jal d_reg
615         nop
616         j       print_regs_out
617         nop
618
619 ### ------
620
621 ### 8: target
622     c_8:
623         la      $a0,    tb
624         li      $v0,    4               # syscall_4: print string
625         syscall
626
627         jal target_field
628         nop
629         j       print_regs_out
630         nop
631 ### ------
632
633 ### 9: rt, rs, imm
634     c_9:
635         la      $a0,    tb
636         li      $v0,    4               # syscall_4: print string
637         syscall
638
639         jal t_reg
640         nop
641
642         la      $a0,    comma
643         li      $v0,    4               # syscall_4: print string
644         syscall
645
646         la      $a0,    tb
647         li      $v0,    4               # syscall_4: print string
648         syscall
649
650         jal s_reg
651         nop
652
653         la      $a0,    comma
654         li      $v0,    4               # syscall_4: print string
655         syscall
656
657         la      $a0,    tb
658         li      $v0,    4               # syscall_4: print string
659         syscall
660
661         jal imm_field
662         nop
663         j       print_regs_out
664         nop
665 ### ------
666
667 ### 10: No fields
668     c_10:
669         j       print_regs_out
670         nop
671 ### ------
672
673 ### 11: rt, imm
```

```
    c_11:
        la      $a0,    tb
        li      $v0,    4               # syscall_4: print string
        syscall

        jal t_reg
        nop

        la      $a0,    comma
        li      $v0,    4               # syscall_4: print string
        syscall

        la      $a0,    tb
        li      $v0,    4               # syscall_4: print string
        syscall

        jal imm_field
        nop
        j       print_regs_out
        nop
### ------

### 12: rt,imm(rs)
    c_12:
        la      $a0,    tb
        li      $v0,    4               # syscall_4: print string
        syscall

        jal t_reg
        nop

        la      $a0,    comma
        li      $v0,    4               # syscall_4: print string
        syscall

        la      $a0,    tb
        li      $v0,    4               # syscall_4: print string
        syscall

        jal imm_field
        nop

        la      $a0,    l_paren
        li      $v0,    4               # syscall_4: print string
        syscall

        jal s_reg
        nop

        la      $a0,    r_paren
        li      $v0,    4               # syscall_4: print string
        syscall

        j       print_regs_out
        nop
### ------


##########################

### SEARCH : takes two base addresses ($t1 and $t2) and searches the first set for one
… that matches the current bitpattern splice ($t0), then returns the corresponding text
    search:
        li      $t3,    0
```

```
737     search_loop:
738         # $t4 : address of the current bitpattern from list in .data
739         # $t5 : current bitpattern from list in .data
740         add     $t4,    $t3,    $t1     # add the offset and base address of opcode
  … bitpatterns
741         lb      $t5,    0($t4)          # load the opcode bitpattern from .data, occupies
  … the last 6 digits of the register
742 
743         beq     $t5, $t0, search_found     # if the opcode from the current instruction
  … matchest the current opcode from .data go to found
744         nop
745 
746         addi    $t3,    $t3,    1   # otherwise increment the offset and...
747         j search_loop               # check again
748         nop
749 
750     search_found:
751         move    $t6,    $t3             # saving the offset, because I just realized I'll
  … need it later
752         li      $t9,    8               # using t9 as an immediate register
753         mult    $t3,    $t9
754         mflo    $t3
755         add     $a0,    $t3,    $t2     # add new offset and opcode string base address
  … and put in a0 to be printed back in print_opc_out NOT IN OPC_OUT, NEED TO CHANGE NAME
756 
757         jr      $ra                 # jump to $ra
758         nop
759 ### ------
760 
761 ### SPLICE BITS : returns bits a0 to a1 (0 index) of the current instruction to t0 as the
  … least significant bits of the register
762     splice_bits:
763         sllv    $t0,    $s1,    $a0
764         li      $t1,    31
765         sub     $a1,    $t1,    $a1
766         add     $a1,    $a1,    $a0
767         srlv    $t0,    $t0,    $a1
768         jr      $ra                 # jump to $ra
769         nop
770 ### ------
771 
772 ### CHECK IF NOP
773     check_if_nop:
774         bne     $s1,    $0, not_nop
775 
776         la      $a0,    the_nop
777         li      $v0,    4               # syscall_4: print string
778         syscall
779 
780         j       was_nop             # jump to was_nop
781         nop
782 
783     not_nop:
784         jr      $ra                     # jump to $ra
785         nop
786 ### ------
787 
788 ### CHECK IF LAST : checks if the current instruction is the last one, if it is it sets
  … s7 to 1, otherwise it returns without changing anythin
789     check_if_last:
790         li      $t0,    0xC             # immediate register : syscall bitpattern
791         bne     $s1,    $t0,    not_last_out
792         nop
793         li      $t0,    0x2402000A      # immediate register : instruction that would
```

```
793… have loaded 10 into $v0
794         lw      $t1,    -4($s0)         # load the last instruction into $t1
795         bne     $t0,    $t1,    not_last_out    # if the last instruction is not the
  … expected one that would load 10 into $v0, then this is not the end
796         nop
797         li      $s7,    1               # a 1 in s7 will indicate to the program that
  … this is the last instruction
798     not_last_out:
799         jr      $ra                     # jump to $ra
800         nop
801 ### ------
802
803 ### GET INST : loads the current instruction into a register
804     get_inst:
805         lw      $s1,    0($s0)          # copy instruction into $s1
806         jr      $ra                     # jump to $ra
807         nop
808 ### ------
809
810 ### PRINT HEX INST : prints the hex of each instruction
811     print_hex_inst:
812         move    $a0,    $s1             # move instruction to be printed
813         li      $v0,    34              # syscall_1: print int
814         syscall
815
816         la      $a0,    tb
817         li      $v0,    4               # syscall_4: print_string
818         syscall
819
820         jr      $ra                     # jump to $ra
821         nop
822 ### ------
823
824 ### END
825     end:
826         li      $v0,    10
827         syscall
828 ### ------
829
```