Assignment 2: Improve one instruction assembler

One instruction assembler (ASS) is a program to convert any string put in the data segment to MIPS32 machine code and run the converted code. However, ASS capacity is very limited. It can convert only one instruction"`jr $ra$`" and it will put 0x0 for other things else.

The implementation is described by state diagram shown in Figure 1.

Run and try out ASS.
And, help improving it.

Tasks

1. Extending ASS register reference.
Extend ASS to be able to take `$s0` as a valid register reference, so that "`jr $s0\n`" will be recognized and get its corresponding machine code.

2. Handling white spaces.
Improve ASS functionality so that the following features are included.
a) White spaces coming first or last in a line do not change the validity of the instruction: ASS should treat "` jr $ra\n`", "`jr $ra \n`", and "` jr $ra \n`" as a valid instruction "`jr $ra`".
b) Multiple white spaces are allowed, e.g. "`jr  $ra\n`" should be as good as "`jr $ra\n`".

3. Find flaws in the state diagram.
Identify at least a sequence of strings that ASS could not work as expected.

B1. * Bonus*: Handling starting character.
A code beginning with "`\n`" or containing "`\n\n`" would make ASS miss the following "`jr $ra\n`". For example, instead of converting "`\njr $ra\n`" to `0x00000000` (for '\n') and `0x03E00008` (for '`jr $ra\n`'), ASS could not recognize the line of nothing ("`\n`") and would just inadvertently put `0x00000000` for "`\njr $ra\n`". Fix this flaw.

B2. * Bonus *: Improving termination.
According to the state diagram, ASS would not be able to recognize the terminating character "`;`" when it occurs in other states but state ="". Fix this so that whenever the terminating character shows up, ASS to go to the termination state.

B3. * Bonus *: Extending ASS to include another instruction.
Extend ASS to be able to recognize another instruction, but not "`jr`" nor "`nop`".

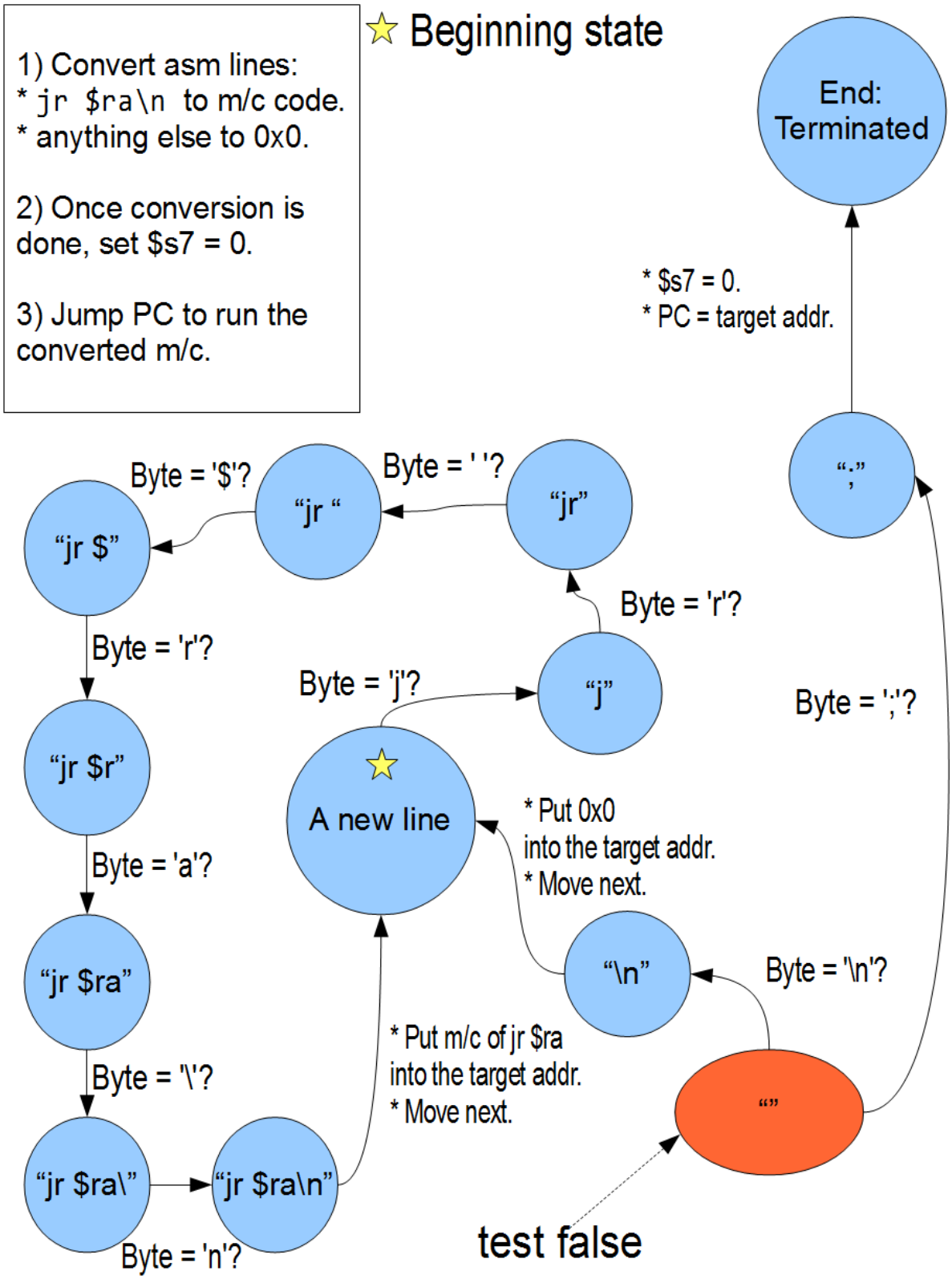

Figure 1: State diagram of ASS

Code of one instruction assembler (for QtSpim)

```
# Program to read a number and print its square
.data

myasm:     .asciiz    "asm \njr $ra\n ISA \njr $ra\nMIPS32;"

#.data 0x10010024 (Program has to be put in address between 0x400000 and
0x10000000. See Fig 3.22 P&H2e)
# mccode: .asciiz          "!!!! The m/c code will be here. !!!!"
# expect to see       "00000000;03E00008;00000000;03E00008

           .globl main
      .text
main:
#     NOP              #START PROGRAM 00First
# initialization
      la $s0, myasm         # $s0 <- base address of myasm
      la $s1, mccode        # $s1 <- base address of mccode
      add $s2, $zero, $zero # $s2 <- byte pointer (increment by 1)
      add $s3, $zero, $zero # $s3 <- output pointer (increment by 4)
      addi $s7, $zero, 0xffffffff     # $s7 <- 0xffffffff

newline:
# State = A new line

      # read byte.
      add $t1, $s0, $s2     # $t1 has current byte address
      addi $s2, $s2, 1# increase the byte pointer
      lb $t0,    0($t1)          # $t0 <- read byte

      # check if byte = 'j'.
      li $t2, 0x6A          # j: 0x6A; State = A new line
      bne $t0, $t2, testfalse

# State = "j"

      # read byte.
      add $t1, $s0, $s2     # $t1 has current byte address
      addi $s2, $s2, 1# increase the byte pointer
      lb $t0,    0($t1)          # $t0 <- read byte

      # check if byte = 'r'.
      li $t2, 0x72          # r: 0x72; State = "j"
      bne $t0, $t2, testfalse
```

```
# State = "jr"

     # read byte.
     add $t1, $s0, $s2    # $t1 has current byte address
     addi $s2, $s2, 1# increase the byte pointer
     lb $t0,    0($t1)          # $t0 <- read byte

     # check if byte = ' '.
     li $t2, 0x20         # <space>: 0x20; State = " "
     bne $t0, $t2, testfalse

# State = "jr "

     # read byte.
     add $t1, $s0, $s2    # $t1 has current byte address
     addi $s2, $s2, 1# increase the byte pointer
     lb $t0,    0($t1)          # $t0 <- read byte

     # check if byte = '$'.
     li $t2, 0x24         # $: 0x24; State = "jr "
     bne $t0, $t2, testfalse

# State = "jr $"

     # read byte.
     add $t1, $s0, $s2    # $t1 has current byte address
     addi $s2, $s2, 1# increase the byte pointer
     lb $t0,    0($t1)          # $t0 <- read byte

     # check if byte = 'r'.
     li $t2, 0x72         # r: 0x72; State = "jr $"
     bne $t0, $t2, testfalse

# State = "jr $r"

     # read byte.
     add $t1, $s0, $s2    # $t1 has current byte address
     addi $s2, $s2, 1# increase the byte pointer
     lb $t0,    0($t1)          # $t0 <- read byte

     # check if byte = 'a'.
     li $t2, 0x61         # a: 0x61; State = "jr $r"
     bne $t0, $t2, testfalse

# State = "jr $ra"
```

```
     # read byte.
     add $t1, $s0, $s2     # $t1 has current byte address
     addi $s2, $s2, 1# increase the byte pointer
     lb $t0,    0($t1)          # $t0 <- read byte

     # check if byte = '\n'.
     li $t2, 0x0A          # \n: 0x0A (Line Feed); State = "jr $ra"
     bne $t0, $t2, testfalse

# State = "jr $ra\n"

     # put m/c for jr $ra into the target address.
     li $t3, 0x03E00008    # 0b0000 0011 1110 0000 0000 0000 0000 1000
     add $t1, $s1, $s3
     addi $s3, $s3, 4
     sw $t3, 0($t1)        # putting m/c code; State = "jr $ra\n"

     # move next
     j newline

testfalse:
# State = "":

     #read byte.
     add $t1, $s0, $s2     # $t1 has current byte address
     addi $s2, $s2, 1# increase the byte pointer
     lb $t0,    0($t1)          # $t0 <- read byte

     # check if byte = '\n'.
     li $t2, 0x0A          # \n: 0x0A (Line Feed); State = ""
     bne $t0, $t2, checkEndChr

# State = "\n":

     # put 0x0 into the target addr.
     add $t3, $zero, $zero # $t3 <- 0
     add $t1, $s1, $s3
     addi $s3, $s3, 4
     sw $t3, 0($t1)        # putting 0x0; State = "\n"

     # move next
     j newline

checkEndChr:
     # check if byte = ';' (Is it ";"?)
     li $t2, 0x3B          # ;: 0x3B; State = ";"
```

```
      bne $t0, $t2, testfalse

# State = "Terminated":

      # $s7 = 0
      add $s7, $zero, $zero

      # PC = target addr.
      jr $s1                # PC = Address of m/c

#     NOP              #END PROGRAM
end:
      li   $v0,10           #System call code to Exit
      syscall #Call OS to Exit the program

mccode:
      nop
      nop              # expect to see jr $ra put here.
      nop
      nop
```