El lenguaje `m2r`

El lenguaje es un ensamblador para una máquina imaginaria (`m2r`), que sólo tiene dos registros, `A` y `B`, que actúan como acumulador y registro base respectivamente. Tiene un espacio de memoria para el código lo suficientemente grande como para almacenar un programa típico, y otro espacio de 16384 posiciones de memoria para datos (de la 0 a la 16383). Este espacio de datos debe ser gestionado por el compilador según el criterio de quien lo diseñe. Los tipos de datos básicos, entero y real, se pueden almacenar en *una* posición de memoria (aunque internamente se representan de distinta manera y por tanto un valor entero no puede utilizarse como si fuera real sin una conversión explícita, por ejemplo).

El intérprete de la máquina objeto entiende texto ASCII compuesto por instrucciones, una por línea, tomadas del conjunto que se describe seguidamente. Las instrucciones pueden ir precedidas de un número de línea, de una etiqueta o de nada. El intérprete admite comentarios que empiezan por '`;`' y terminan con el final de la línea.

Instrucciones

`mov` *fuente destino*	Copia en *destino* el valor *fuente*.
`addi` *fuente*	Suma a `A` el valor *fuente*. Se supone que tanto el acumulador como el valor *fuente* son enteros, y el resultado es también entero.
`addr` *fuente*	Igual que `addi`, pero todos los operandos y el resultado son reales.
`subi` *fuente*	Resta de `A` (entero) el valor *fuente* (entero).
`subr` *fuente*	Igual que `subi` pero con reales.
`muli` *fuente*	Multiplica `A` (entero) por el valor *fuente* (entero) y guarda el resultado en `A`.
`mulr` *fuente*	Igual que `muli` pero con reales.
`divi` *fuente*	Divide `A` (entero) por el valor *fuente* (entero) y guarda el cociente de la división entera en `A`.
`divr` *fuente*	Igual que `divi` pero con reales.
`modi` *fuente*	Igual que `divi`, pero en lugar de guardar el cociente guarda el resto de la división.
`andi` *fuente*	Deja en `A` un 1 si el valor (entero) de `A` y el valor (entero) de *fuente* son iguales a 1, y deja un 0 en otro caso.
`andr` *fuente*	Igual que `andi` pero se supone que `A` y *fuente* son reales, aunque el resultado (que queda en `A`) es entero.
`ori` *fuente*	Deja en `A` un 0 si el valor (entero) de `A` y el valor (entero) de *fuente* son iguales a 0, y deja un 1 en otro caso.
`orr` *fuente*	Igual que `ori` pero se supone que `A` y *fuente* son reales, aunque el resultado (que queda en `A`) es entero.
`noti`	Deja en `A` un 1 si el valor (entero) de `A` es cero, y deja un 0 en otro caso.
`notr`	Igual que `noti`, pero el valor de `A` debe ser real y el resultado es entero.
`itor`	Convierte el valor entero de `A` en un valor real y lo deja en el acumulador.
`rtoi`	Trunca el valor real de `A` al entero más cercano y lo deja en el acumulador.
`halt`	Detiene la máquina (el intérprete).
`wri` *fuente*	Imprime el valor (entero) de *fuente*.
`wrr` *fuente*	Imprime el valor (real) de *fuente*.
`wrc` *fuente*	Imprime el carácter representado por los 8 bits más bajos del valor entero *fuente*.

`wrl`	Imprime un salto de línea.
`rdi` *destino*	Lee un entero de la consola y lo carga en *destino*.
`rdr` *destino*	Lee un real de la consola y lo carga en *destino*.
`rdc` *destino*	Lee un carácter de la consola y carga su código ASCII en *destino*.
`eqli` *fuente*	Deja un 1 en `A` si el valor (entero) de `A` es igual que el valor (entero) *fuente*, y deja un 0 en otro caso.
`eqlr` *fuente*	Igual que `eqli`, pero los operandos son reales y el resultado entero.
`neqi` *fuente*	Deja un 1 en `A` si el valor (entero) de `A` es distinto del valor (entero) *fuente*, y deja un 0 en otro caso.
`neqr` *fuente*	Igual que `neqi`, pero los operandos son reales y el resultado entero.
`gtri` *fuente*	Deja un 1 en `A` si el valor (entero) de `A` es mayor que el valor (entero) *fuente*, y deja un 0 en otro caso.
`gtrr` *fuente*	Igual que `gtri`, pero los operandos son reales y el resultado entero.
`geqi` *fuente*	Deja un 1 en `A` si el valor (entero) de `A` es mayor o igual que el valor (entero) *fuente*, y deja un 0 en otro caso.
`geqr` *fuente*	Igual que `geqi`, pero los operandos son reales y el resultado entero.
`lssi` *fuente*	Deja un 1 en `A` si el valor (entero) de `A` es menor que el valor (entero) *fuente*, y deja un 0 en otro caso.
`lssr` *fuente*	Igual que `lssi`, pero los operandos son reales y el resultado entero.
`leqi` *fuente*	Deja un 1 en `A` si el valor (entero) de `A` es menor o igual que el valor (entero) *fuente*, y deja un 0 en otro caso.
`leqr` *fuente*	Igual que `leqi`, pero los operandos son reales y el resultado entero.
`jmp` *posprog*	Salta a la posición de programa indicada por el valor *posprog*.
`jz` *posprog*	Salta a la posición de programa indicada por el valor *posprog* si en `A` hay un cero. El valor de `A` debe ser entero.
`jnz` *posprog*	Salta a la posición de programa indicada por el valor *posprog* si en `A` hay un número entero distinto de cero.
`mvetq` *etiqueta destino*	Copia en *destino* la posición de programa asociada a la *etiqueta*. Esta instrucción sirve para almacenar la posición de programa a la que hay que volver después de una llamada a una función.

Las clases posibles de *destino* son:

`n`	la dirección de memoria `n`.
`A`	el acumulador.
`@A`	la dirección de memoria que representa el valor que hay en `A`.
`@B+n`	la dirección que se obtiene de sumar `n` al contenido de `B`.
`@B-n`	la dirección que se obtiene de restar `n` al contenido de `B`.
`B`	el registro base.

Las clases posibles de *fuente* son:

`#i` el valor numérico entero `i`.

`$r` el valor numérico real `r`.

`n` el valor almacenado en la dirección de memoria `n`.

`A` el valor almacenado en el acumulador.

`@A` el valor almacenado en la dirección que representa el valor que hay en `A`.

`@B+n` el valor almacenado en la dirección que se obtiene de sumar `n` al contenido de `B`.

`@B-n` el valor almacenado en la dirección que se obtiene de restar `n` al contenido de `B`.

`B` el valor almacenado en el registro base.

Las clases posibles de *posprog* son:

`n` la posición de programa `n`.

`Ln` una etiqueta, compuesta por la letra `L` seguida de un número: por ejemplo, `L25`.

`@A` la posición de programa contenida en `A`.

`@B+n` la posicion de programa contenida en la dirección que se obtiene de sumar `n` al contenido de `B`.

`@B-n` la posicion de programa contenida en la dirección que se obtiene de restar `n` al contenido de `B`.

Un ejemplo de programa objeto correcto (aunque no necesariamente traducción de un programa fuente determinado) sería el siguiente:

```
mov #2 A     ; guarda 2 en A
addi #3      ; suma 3 a A
mov A 23     ; guarda el valor de A (5) en la direccion 23
mov 23 A     ; guarda el contenido de la direccion 23 (5) en A
subi #3      ; resta 3 de A
wri A        ; imprime el valor almacenado en A (2)
wrl          ; imprime un salto de linea
mov #7 A     ; guarda 7 en A
itor         ; convierte el 7 que en A en real (7.0)
divr $3.5    ; divide el valor que hay en A (7.0) por 3.5
wrr A        ; imprime el valor real que hay en A (2.0)
wrl          ; imprime un salto de linea
halt         ; termina el programa
```

El texto que sigue a '`;`' en cada línea es un comentario.