Proyecto de Programación en Ensamblador
Estructura de Computadores
Grado en Ingeniería Informática

Ejemplos de casos de prueba

Departamento de Arquitectura y Tecnología de Sistemas Informáticos

2024-2025

Ejemplos

Este documento complementa la sección "Ejemplos" del enunciado del proyecto de programación en ensamblador (pág. 24). Contiene varios ejemplos de casos de prueba para cada una de las subrutinas que componen el proyecto.

Estos ejemplos no constituyen un juego de ensayo completo. No sirven por sí solos para comprobar el correcto funcionamiento de las subrutinas ni para detectar de forma completa en qué situaciones presentan comportamientos erróneos. Sin embargo, se pueden seguir como guía para la elaboración de juegos de ensayo más completos.

En cada ejemplo se presenta el contenido de la memoria principal antes y después de la ejecución de una subrutina. Este contenido se muestra tal y como lo hace el simulador del 88110 usando el comando "V". En concreto, se muestra lo siguiente:

- Los datos de prueba suministrados como parámetros a las subrutinas.
- El contenido de la pila al comienzo de su ejecución.
- Los resultados que producen.

En algunos casos también se describen los datos de prueba en lenguaje ensamblador. En otros en su descripción se indica "Los registros parten de valores distintos de 0"; en estos casos, antes de la llamada a la subrutina, se inicializan los registros con valores indefinidos.

Por último, en la página 13, se incluye el programa de prueba de un caso concreto para la subrutina `BuscaMax`.

En este procesador el direccionamiento se hace a nivel de byte y se utiliza el formato *little-endian*. En consecuencia, cada una de las palabras representadas a continuación de la especificación de la dirección debe interpretarse como formada por 4 bytes con el orden que se muestra en el ejemplo siguiente:

```
Direcciones de memoria, tal como las representa el simulador:
60000       04050607     05010000

Direcciones de memoria, tal como se deben interpretar:
60000       04
60001       05
60002       06
60003       07

60004       05
60005       01
60006       00
60007       00

Valor de las palabras almacenadas en las posiciones 60000 y
60004, tal como la interpreta el procesador:

60000       0x07060504 = 117.835.012
60004       0x00000105 = 261
```

Longitud de Cadena

Caso 1. Llamada a `LongCad`

Llama a `LongCad` pasándole como parámetro la cadena `"Prueba\0"` con `r29` inicializado a `100`.

```
Cadena:
"Prueba\0"
```

Inicial. Registros: `r30=86012 (0x14FFC) r29=100 (0x64)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ffc 1
     86000                                                  00200100
88110> v 0x12000 2
     73728       50727565     62610000
```

Resultado. Registros: `r30=86012 (0x14FFC) r29=6 (0x06)`

Caso 2. Llamada a `LongCad`

Llama a `LongCad` pasándole como parámetro la cadena `"123456789 123456789 12\0A"`

```
Cadena:
"123456789 123456789 12\0A"
```

Inicial. Registros: `r30=86012 (0x14FFC)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ffc 1
     86000                                                  00200100
88110> v 0x12000 7
     73728       31323334     35363738     39203132     33343536
     73744       37383920     31320041     A55A0000
```

Resultado. Registros: `r30=86012 (0x14FFC) r29=22 (0x16)`

Búsqueda de carácter

Caso 3. Llamada a `BuscaCar`

Llama a `BuscaCar`, pasándole una cadena de 10 caracteres y trata de localizar el situado en la posición 0 y en otras. El valor del parámetro `from` es mayor que cero

```
Cadena:
"*2345*78*0\0"
```

Inicial. Registros: `r30=86000 (0x14FF0)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff0 4
     86000       2A000000     00200100     04000000     0A000000
88110> v 0x12000 3
     73728       2A323334     352A3738     2A300000
```

Resultado. Registros: `r30=86000 (0x14FF0) r29=5 (0x05)`

Caso 4. Llamada a `BuscaCar`

Llama a `BuscaCar`, pasándole una cadena de 10 caracteres y trata de localizar un carácter que no está presente

```
Cadena:
"*2345*78*0\0"
```

Inicial. Registros: `r30=86000 (0x14FF0)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff0 4
     86000       31000000     00200100     04000000     0A000000
88110> v 0x12000 3
     73728       2A323334     352A3738     2A300000
```

Resultado. Registros: `r30=86000 (0x14FF0) r29=10 (0x0A)`

Caso 5. Llamada a `BuscaCar`

Llama a `BuscaCar`, pasándole una cadena de 10 caracteres y trata de localizar uno de ellos cuyo valor numérico es mayor de 150 y está situado hacia el final de la cadena

```
Cadena (definida numéricamente):
0x40302010, 0x80706050, 0xA090
```

Inicial. Registros: `r30=86000 (0x14FF0)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff0 4
     86000       A0000000     00200100     04000000     0A000000
88110> v 0x12000 3
     73728       10203040     50607080     90A00000
```

Resultado. Registros: `r30=86000 (0x14FF0) r29=9 (0x09)`

Coincidencia de cadenas

Caso 6. Llamada a `CoincidenCad`

Llama a `CoincidenCad`, pasándole dos cadenas de 10 caracteres que comienzan por un carácter diferente. Todos los registros parten de valores iniciales distintos de cero

```
Cadena1 ... Cadena2:
"1234567890\0" ... "0123456789\0"
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                   00200100     10200100
88110> v 0x12000 3
     73728       31323334     35363738     39300000
88110> v 0x12010 3
     73744       30313233     34353637     38390000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=0 (0x0)`

Caso 7. Llamada a `CoincidenCad`

Llama a `CoincidenCad`, pasándole dos cadenas de tamaño mediano en las que coinciden cerca de la primera mitad de sus caracteres

```
Cadena1:
"12*3456789012*3456789012*3456789012*3456789012*3456789012*34567890\0"
Cadena2:
"12*3456789012*3456789012*345678901*3456789012*3456789012*34567890\0"
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                   00200100     48200100
88110> v 0x12000 17
     73728       31322A33     34353637     38393031     322A3334
     73744       35363738     39303132     2A333435     36373839
     73760       3031322A     33343536     37383930     31322A33
     73776       34353637     38393031     322A3334     35363738
     73792       39300000
88110> v 0x12048 17
     73792                                   31322A33     34353637
     73808       38393031     322A3334     35363738     39303132
     73824       2A333435     36373839     30312A33     34353637
     73840       38393031     322A3334     35363738     39303132
     73856       2A333435     36373839     30000000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=34 (0x22)`

Activación de un único bit (puesta a 1)

Caso 8. Llamada a `PoneBitA1`

Llama a `PoneBitA1`, pasándole un mapa de bits de solo 1 Byte con todos sus bits a cero y un número de bit intermedio

```
Mapa-de-bits:
0x00 (binario: 0000 0000)
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                  00200100      03000000
88110> v 0x12000 1
     73728       00FFFFFF
```

Resultado. Registros: `r30=86008 (0x14FF8)`
Resultado. Direcciones de memoria:

```
88110> v 0x12000 1
     73728       10FFFFFF
```

Caso 9. Llamada a `PoneBitA1`

Llama a `PoneBitA1` dos veces, pasándole un mapa de bits de solo 1 Byte con todos sus bits a cero (se trata de dos llamadas consecutivas, en cada una de las cuales se pone a 1 un bit diferente)

```
Mapa-de-bits:
0x00 (binario: 0000 0000)
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria (1ª llamada):

```
88110> v 0x14ff8 2
     86000                                  00200100      03000000
88110> v 0x12000 1
     73728       00FFFFFF
```

Inicial. Direcciones de memoria (2ª llamada):

```
88110> v 0x14ff8 2
     86000                                  00200100      05000000
88110> v 0x12000 1
     73728       10FFFFFF
```

Resultado. Registros: `r30=86008 (0x14FF8)`
Resultado. Direcciones de memoria:

```
88110> v 0x12000 1
     73728       14FFFFFF
```

Compresión de texto

Caso 10. Llamada a `Comprime`

Llama a `Comprime`, pasándole una cadena de 10 caracteres que no admite compresión

```
Cadena:
"0123456789\0"
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                   00200100     00210100
88110> v 0x12000 3
     73728       30313233     34353637     3839005A
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=16 (0x10)`
Resultado. Direcciones de memoria:

```
88110> v 0x12100 4
     73984       0A000106     00003031     32333435     36373839
```

Caso 11. Llamada a `Comprime`

Llama a `Comprime`, pasándole una cadena de caracteres igual a la del ejemplo detallado en la presentación del proyecto (N=4)

```
Cadena:
"tres tristes tigres comen trigo en un trigal, el primer tigre que...\0"
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                   00200100     00210100
88110> v 0x12000 18
     73728       74726573     20747269     73746573     20746967
     73744       72657320     636F6D65     6E207472     69676F20
     73760       656E2075     6E207472     6967616C     2C20656C
     73776       20707269     6D657220     74696772     65207175
     73792       652E2E2E     00000000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=70 (0x46)`
Resultado. Direcciones de memoria:

```
88110> v 0x12100 18
     73984       4400010B     00241010     00400074     72657320
     74000       74726973     74020004     69670100     04636F6D
     74016       656E0400     04676F20     656E2075     18000661
     74032       6C2C2065     6C207072     696D6572     0C000620
     74048       7175652E     2E2E0000
```

Lectura de un único bit

Caso 12. Llamada a `LeeBit`

Llama a `LeeBit`, pasándole un mapa de bits de un único Byte con tres bits a 1 y el resto a 0

```
Mapa-de-bits:
0x54 (binario: 0101 0100)
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                  00200100      03000000
88110> v 0x12000 1
     73728       54000000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=1 (0x00000001)`

Caso 13. Llamada a `LeeBit`

Llama a `LeeBit` dos veces, pasándole un mapa de bits de un único Byte con tres bits a 1 y el resto a 0 (se trata de dos llamadas consecutivas, en cada una de las cuales se lee un bit diferente)

```
Mapa-de-bits:
0x54 (binario: 0101 0100)
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria (1ª llamada):

```
88110> v 0x14ff8 2
     86000                                  00200100      03000000
88110> v 0x12000 1
     73728       54000000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=1 (0x00000001)`
Inicial (2ª llamada). Direcciones de memoria (2ª llamada):

```
88110> v 0x14ff8 2
     86000                                  00200100      04000000
88110> v 0x12000 1
     73728       54000000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=0 (0x00000000)`

Descompresión de texto

Caso 14. Llamada a `Descomprime`

Llama a `Descomprime`, pasándole un texto comprimido que corresponde a una cadena de 20 caracteres que no admite compresión

```
Definición del texto comprimido de entrada:
      org     0x12000
CMPR: data    0x07010014, 0x30000000, 0x34333231, 0x38373635
      data    0x37383939, 0x33343536, 0x00303132

Cadena resultante en ASCII:
"01234567899876543210\0"
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                  00200100     00210100
88110> v 0x12000 7
     73728       14000107     00000030     31323334     35363738
     73744       39393837     36353433     32313000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=20 (0x14)`
Resultado. Direcciones de memoria:

```
88110> v 0x12100 6
     73984       30313233     34353637     38393938     37363534
     74000       33323130     00000000
```

Caso 15. Llamada a `Descomprime`

Llama a `Descomprime`, pasándole un texto comprimido que corresponde a una cadena de caracteres igual a la del ejemplo detallado en la presentación del proyecto (N=4)

```
Definición del texto comprimido de entrada:
      org     0x12000
CMPR: data    0x0b010044, 0x10102400, 0x74004000, 0x20736572
      data    0x73697274, 0x04000274, 0x00016769, 0x6d6f6304
      data    0x00046e65, 0x206f6704, 0x75206e65, 0x61060018
      data    0x65202c6c, 0x7270206c, 0x72656d69, 0x2006000c
      data    0x2e657571, 0x00002e2e


Cadena resultante en ASCII:
"tres tristes tigres comen trigo en un trigal, el primer tigre que...\0"
```

Inicial. Registros: `r30=86008 (0x14FF8)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff8 2
     86000                                  00200100     00210100
88110> v 0x12000 18
     73728       4400010B     00241010     00400074     72657320
     73744       74726973     74020004     69670100     04636F6D
     73760       656E0400     04676F20     656E2075     18000661
     73776       6C2C2065     6C207072     696D6572     0C000620
     73792       7175652E     2E2E0000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29=68 (0x44)`
Resultado. Direcciones de memoria:

```
88110> v 0x12100 18
     73984       74726573     20747269     73746573     20746967
     74000       72657320     636F6D65     6E207472     69676F20
     74016       656E2075     6E207472     6967616C     2C20656C
     74032       20707269     6D657220     74696772     65207175
     74048       652E2E2E     00000000
```

Verificación de resultados

Caso 16. Llamada a `Verifica`

Llama a `Verifica`, pasándole una cadena de 10 caracteres que no admite compresión

```
Cadena:
"0123456789\0"
```

Inicial. Registros: `r30=86004 (0x14FF4)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff4 3
     86000                    00200100     00220100     04220100
88110> v 0x12000 3
     73728       30313233     34353637     3839005A
```

Resultado. Registros: `r30=86008 (0x14FF8) r29 = 0 (0x0)`
Resultado. Direcciones de memoria:

```
88110> v 0x12200 2
     74240       0A000000     0A000000
```

Caso 17. Llamada a `Verifica`

Llama a `Verifica`, pasándole una cadena de caracteres similar a la del ejemplo detallado en la presentación del proyecto

```
Cadena:
"tres tristes tigres comen trigo en un trigal, el primer tigre que...\0"
```

Inicial. Registros: `r30=86004 (0x14FF4)`
Inicial. Direcciones de memoria:

```
88110> v 0x14ff4 3
     86000                    00200100     00220100     04220100
88110> v 0x12000 18
     73728       74726573     20747269     73746573     20746967
     73744       72657320     636F6D65     6E207472     69676F20
     73760       656E2075     6E207472     6967616C     2C20656C
     73776       20707269     6D657220     74696772     65207175
     73792       652E2E2E     00000000
```

Resultado. Registros: `r30=86008 (0x14FF8) r29 = 0 (0x0)`
Resultado. Direcciones de memoria:

```
88110> v 0x12200 2
     74240       44000000     44000000
```

```
;
; Llama a 'BuscaMax', pasándole una cadena de 10 caracteres y
; una posición tal que la subcadena buscada tiene un tamaño
; máximo igual a tres caracteres
;
    org 0x8000

CAD1:   data    "0123789789\0"
        data    0x55AA  ; valor arbitrario
JJ:     data    0x5AA5  ; valor arbitrario

    org 0x8400

ppal:   LEA (r30, 0x0F000)
        or      r31, r30, r30

        LEA     (r10, CAD1)     ; cad
        addu    r11, r0, 7      ; max=7
        LEA     (r12, JJ)       ; jj

        PUSH    (r12)
        PUSH    (r11)
        PUSH    (r10)

        bsr     BuscaMax

        addu    r30, r30, 12    ; devuelve r30 al valor inicial

        stop
```

Figura 1. Programa de prueba de BuscaMax.