Lista 5

tags: ASK

Bitowa ściągawka

Flagi

- CF - carry flag
- OF - overflow flag
- PF - parity flag
- SF - sign flag
- ZF - zero flag
- AF - aux

Argumenty f(%rdi, %rsi, %rdx, %rcx, %r8, %r9)

Rejestry

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b

Zadanie 1

Zadanie 1. Zapisz w języku `C` funkcję o sygnaturze «`int puzzle(long x, unsigned n)`» której kod w asemblerze podano niżej. Zakładamy, że parametr «`n`» jest niewiększy niż 64. Przedstaw jednym zdaniem co robi ta procedura.

```
Dokumentacja dla testl SRC1, SRC2

TEMP ← SRC1 AND SRC2;
SF ← MSB(TEMP);
IF TEMP = 0
    THEN ZF ← 1;
    ELSE ZF ← 0;
FI:
PF ← BitwiseXNOR(TEMP[0:7]);
CF ← 0;
OF ← 0;
(* AF is undefined *)
```

puzzle(long x, unsigned n)

```
puzzle: testl %esi, %esi //esi == 0 ? ZF = 1 : ZF = 0
        je .L4           //ZF == 1 ? jump
        xorl %edx, %edx  //edx = 0
        xorl %eax, %eax  //eax = 0
.L3:    movl %edi, %ecx  //ecx = edi
        andl $1, %ecx    //ecx = LSB(ecx)
        addl %ecx, %eax  //eax += ecx
        sarq %rdi        //rdi = rdi >> 1
        incl %edx        //edx += 1
        cmpl %edx, %esi  //esi - edx == 0 ? ZF = 1 : ZF = 0
        jne .L3          //ZF == 0 ? jump
        ret
.L4:    movl %esi, %eax
        ret
```

Funkcja kończy się natychmiast z wynikiem 0, jeśli x = 0. W przeciwnym przypadku w każdym obrocie pętli wyłuskiwany jest ostatni bit x i dodawany do wyniku. Następnie x jest shiftowany w prawo. Ten proces jest powtarzany n razy.

Jednym zdaniem funkcja puzzle zlicza ostatnie n bitów x.

```
int puzzle(long x, unsigned int n) {
    int res = 0;

    for(unsigned int i = 0; i < n; i++) {
        res += x & 1;
        x = x >> 1;
    }
}
```

Zadanie 2

Zadanie 2. Poniżej zamieszczono kod procedury o sygnaturze «`long puzzle2(char *s, char *d)`». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania. Przetłumacz tę procedurę na język C, a następnie jednym zdaniem powiedz co ona robi.

puzzle2(char *s, chard *d)

```
puzzle2:
          movq %rdi, %rax    //rax = s
.L3:      movb (%rax), %r9b  //r9b = *s
          leaq 1(%rax), %r8  //r8 = s + 1
          movq %rsi, %rdx    //rdx = d
.L2:      movb (%rdx), %cl   //cl = *d
          incq %rdx          //rdx += 1
          testb %cl, %cl     //*d == 0 ? ZF = 1 : ZF = 0
          je .L4             //ZF == 1 ? jump
          cmpb %cl, %r9b     //*s - *d == 0 ? ZF = 1: ZF = 0
                             //(czyli znaki jeśli są równe to ZF = 1)
          jne .L2            //ZF == 0 ? jump
          movq %r8, %rax     //rax = s + 1
          jmp .L3
.L4:      subq %rdi, %rax    //rax -= s
          ret
```

Funkcja znajduje taki prefix ciągu *s, że dla każdego znaku w *s istnieje taki sam znak w *d.

```
long puzzle(char *s, char *d) {

    char *siter = s;
    while(1) {
        char *diter = d;
        while(1) {
            if (*diter == 0) {return siter - s;}
            if (*diter == *siter) {
                s++;
                break;
            }
            diter++;
        }
    }

}
```

Bloki

```
puzzle2:
          movq %rdi, %rax    //B1
.L3:      movb (%rax), %r9b  //B2
          leaq 1(%rax), %r8
          movq %rsi, %rdx
.L2:      movb (%rdx), %cl   //B3
          incq %rdx
          testb %cl, %cl
          je .L4
          cmpb %cl, %r9b     //B4
          jne .L2
          movq %r8, %rax     //B5
          jmp .L3
.L4:      subq %rdi, %rax    //B6
          ret
```

```
digraph {
    Start -> B1
    B1 -> B2
    B2 -> B3
    B3 -> B4
    B3 -> B6
    B4 -> B3
    B4 -> B5
    B5 -> B2
    B6 -> Stop
}
```

Zadanie 3

Zadanie 3 (2). Poniżej widnieje kod funkcji o sygnaturze «`uint32_t puzzle3(uint32_t n, uint32_t d)`». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania, po czym przetłumacz tę funkcję na język C. Na podstawie ustępu „Mixing C and Assembly Language" strony GNU Assembler Examples[4] napisz i zaprezentuj działanie programu, który pomógł Ci powiedzieć co ta funkcja robi.

puzzle3(uint32_t n, uint32_t d)

```
puzzle3:
        movl %edi, %edi         //piszemy do mniejszych bitów, więc zerujemy starsze
        salq $32, %rsi          //d = d << 32 (shift arytmetyczny)
        movl $32, %edx          //edx = 32 (licznik pętli)
        movl $0x80000000, %ecx  //ecx = 0x80000000 (maska)
        xorl %eax, %eax         //czyścimy eax (tutaj będzie przechowywany wynik)

        //pętla
.L3:    addq %rdi, %rdi         //n *= 2
        movq %rdi, %r8          //r8 = n
        subq %rsi, %r8          // r8 -= d
        js   .L2                //r8 < 0 ? jump L2
                                //(jeśli edi >= d)
        orl  %ecx, %eax            //eax |= ecx
        movq %r8, %rdi             //n = r8

.L2:    shrl %ecx               //ecx = ecx >> 1 (shift logiczny)
        decl %edx               //edx-- (zmniejszamy licznik)
        jne  .L3                //jeśli licznik nie jest zerem to kolejny obrót pętli
        ret
```

```
//n - licznik, d - mianownik
uint32_t puzzle3(uint32_t n, uint32_t d){
    uint64_t num = n;
    uint64_t den = (uint64_t)d << 32;
    uint32_t edx = 32;
    uint32_t mask = 0x80000000;
    uint32_t result = 0;


    while(edx) {
        num *= 2;
        if(num >= den){
            result |= mask;
            num = num - den;
        }
        mask >>= 1;
        edx--;
    }
    return result;
}
```

Bloki

```
puzzle3:                    //B1
        movl %edi, %edi
        salq $32, %rsi
        movl $32, %edx
        movl $0x80000000, %ecx
        xorl %eax, %eax
.L3:    addq %rdi, %rdi      //B2
        movq %rdi, %r8
        subq %rsi, %r8
        js   .L2
        orl  %ecx, %eax      //B3
        movq %r8, %rdi
.L2:    shrl %ecx            //B4
        decl %edx
        jne  .L3
        ret                  //B5
```

```
digraph{
    Start -> B1
    B1 -> B2
    B2 -> B3
    B2 -> B4
    B3 -> B4
    B4 -> B2
    B4 -> B5
    B5 -> Stop
}
```

Zadanie 4

Zadanie 4 (2). Poniżej zamieszczono kod rekurencyjnej procedury o sygnaturze «`int puzzle4(long *a, long v, uint64_t s, uint64_t e)`». Wyznacz bloki podstawowe oraz narysuj graf przepływu sterowania. Przetłumacz tę procedurę na język C, a następnie jednym zdaniem powiedz co ona robi.

```
puzzle4:                        //rax to mid binsearcha
      movq %rcx, %rax           //rax = e
      subq %rdx, %rax           //rax = e - s
      shrq %rax                 //rax = (e-s)/2
      addq %rdx, %rax           //rax = (e-s)/2 + s
      cmpq %rdx, %rcx           //rcx > rdx ? CF = 1
      jb .L5                    //CF == 1 ? jump L5
      movq (%rdi,%rax,8), %r8   //r8 = a[rax] (pobieramy a[mid] do r8)
      cmpq %rsi, %r8            //v == r8 ? ZF == 1 (jeśli a[mid] to szukany element kończymy)
      je .L10                   //ZF == 1 ? jump L10
      cmpq %rsi, %r8            // ... (jeśli v jest większy to sprawdzamy prawą połowę wpp. lewą )
      jg .L11                   //rsi > r8 ? jump L11
      leaq 1(%rax), %rdx        //s = rax + 1
      call puzzle4
.L10: ret                       //e = rax - 1
.L11: leaq -1(%rax), %rcx
      call puzzle4
      ret
.L5:  movl $-1, %eax            //eax = -1
      ret
```

Bloki

```
puzzle4:
      movq %rcx, %rax           //B1
      subq %rdx, %rax
      shrq %rax
      addq %rdx, %rax
      cmpq %rdx, %rcx
      jb .L5
      movq (%rdi,%rax,8), %r8   //B2
      cmpq %rsi, %r8
      je .L10                   //B3
      cmpq %rsi, %r8
      jg .L11                   //B4
      leaq 1(%rax), %rdx
      call puzzle4
.L10: ret                       //B5
.L11: leaq -1(%rax), %rcx       //B6
      call puzzle4
      ret
.L5:  movl $-1, %eax            //B7
      ret
```

```
digraph {
    Start -> B1
    B1 -> B2
    B1 -> B7
    B2 -> B3
    B2 -> B5
    B3 -> B4
    B3 -> B6
    B4 -> B5
    B5 -> Stop
    B6 -> Stop
    B7 -> Stop
}
```

```
int puzzle4(int a*, long v, uint64_t s, uint64_t e)
{
    if (r >= l) {
        int mid = s + (e - s) / 2;

        if (a[mid] == x)
            return mid;

        if (a[mid] > x)
            return puzzle4(a, v, s, mid - 1);

        return puzzle4(a, v, mid + 1, e);
    }

    return -1;
}
```