Introduction to IA-32 Assembly Language Programming - II

Example – power.c

- What it does
  - Reads two integers (x and y)
  - Computes x to the y power
  - Prints the result

```
/*-------------------------------------------------------------*/
/* power.c                                                     */
/* Author: Bob Dondero                                         */
/*-------------------------------------------------------------*/

#include <stdio.h>

static int iBase;
static int iExp;
static int iPower = 1;
static int iIndex;

int main(void)

/* Read a non-negative base and exponent from stdin.  Write base
   raised to the exponent power to stdout.  Return 0. */

{
  printf("Enter the base:  ");
  scanf("%d", &iBase);

  printf("Enter the exponent:  ");
  scanf("%d", &iExp);

  for (iIndex = 1; iIndex <= iExp; iIndex++)
    iPower *= iBase;

  printf("%d to the %d power is %d.\n", iBase, iExp, iPower);

  return 0;
}
```

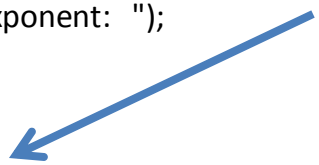
Will be stored in DATA section

This is the problem.
How can we convert this to assembly language?

Refer to “Flattened C Programs” Doc

```
#include <stdio.h>

static int iBase;
static int iExp;
static int iPower = 1;
static int iIndex;

int main(void)

/* Read a non-negative base and exponent from stdin.  Write base
   raised to the exponent power to stdout.  Return 0. */

{
  printf("Enter the base:  ");
  scanf("%d", &iBase);

  printf("Enter the exponent:  ");
  scanf("%d", &iExp);

  iIndex = 1;
loop1:
  if (iIndex > iExp) goto loopend1;
  iPower *= iBase;
  iIndex++;
  goto loop1;
loopend1:

  printf("%d to the %d power is %d.\n", iBase, iExp, iPower);

  return 0;
}
```

```
for (iIndex = 1; iIndex <= iExp; iIndex++)
    iPower *= iBase;
```



```
 .section ".text"
…
main:
…
loop1:
        ## if (iIndex > iExp) goto loopend1
        movl    iIndex, %eax
        cmpl    iExp, %eax
        jg      loopend1

        ## iPower *= iBase
        movl    iPower, %eax
        imull   iBase
        movl    %eax, iPower

        ## iIndex++
        incl    iIndex

        ## goto loop1
        jmp     loop1

loopend1:
…
```

EAX == iIndex

Compare iExp and EAX. Note the order of src and dest

jg means 'Jump greater than' (Conditional Jump)
⇔Jump to loopend if EAX is greater than iExp
⇔Jump to loopend if iIndex is greater than iExp

Unconditional jump to loop1

Control Transfer

- C has data types
  - We define data types (signed / unsigned)
- Assembly language doesn't have data types
  - But different instructions
- It's your responsibility
  - to keep track of the types of data in memory (or register)
  - to use proper instructions based on the types

Control Transfer

- You should use different instructions for
  - multiplication / division
  - control transfer (jump instructions..)

Control Transfer

- Multiplication & Division
  - signed: imul, idiv
  - unsigned: mul, div
- Control Transfers
  - Signed: cmp + [je, jne, jl, jle, jg, jge]
  - Unsigned: (unsigned version of cmp) + [je, jne, jl, jle, jg, jge]
  - Unsigned: cmp + [je, jne, jb, jbe, ja, jae]

Example – powerunsigned.c

- What it does
  - The same as ‘power.c’
  - But uses unsigned ints instead of ints for greater range
- Let’s look at ‘power.c’ in your handouts

```
#include <stdio.h>

static unsigned int uiBase;
static unsigned int uiExp;
static unsigned int uiPower = 1;
static unsigned int uiIndex;

int main(void)

/* Read a non-negative base and exponent from stdin.  Write base
   raised to the exponent power to stdout.  Return 0. */

{
   printf("Enter the base:  ");
   scanf("%u", &uiBase);

   printf("Enter the exponent:  ");
   scanf("%u", &uiExp);

   for (uiIndex = 1; uiIndex <= uiExp; uiIndex++)
      uiPower *= uiBase;

   printf("%u to the %u power is %u.\n", uiBase, uiExp, uiPower);

   return 0;
}
```

- We first convert the file to the flattened version
- Let’s look at the ‘powerunsignedflat.c’

```c
#include <stdio.h>

static unsigned int uiBase;
static unsigned int uiExp;
static unsigned int uiPower = 1;
static unsigned int uiIndex;

int main(void)
{
  printf("Enter the base:  ");
  scanf("%u", &uiBase);

  printf("Enter the exponent:  ");
  scanf("%u", &uiExp);

  uiIndex = 1;
loop1:
  if (uiIndex > uiExp) goto loopend1;
  uiPower *= uiBase;
  uiIndex++;
  goto loop1;
loopend1:

  printf("%u to the %u power is %u.\n", uiBase, uiExp, uiPower);

  return 0;
}
```

```c
for (uiIndex = 1; uiIndex <= uiExp; uiIndex++)
    uiPower *= uiBase;
```

- Let’s look at the ‘powerunsigned.s’

```
...
### ------------------------------------------------------------

        .section ".data"

uiPower:
        .long   1

### ------------------------------------------------------------

        .section ".bss"

uiBase:
        .skip   4

uiExp:
        .skip   4

uiIndex:
        .skip   4

### ------------------------------------------------------------
...
```

```
...

## uiIndex = 1
        movl    $1, uiIndex

loop1:

        ## if (uiIndex > uiExp) goto loopend1
        movl    uiIndex, %eax
        cmpl    uiExp, %eax
        ja      loopend1

        ## uiPower *= uiBase
        movl    uiPower, %eax
        mull    uiBase
        movl    %eax, uiPower

        ## uiIndex++
        incl    uiIndex

        ## goto loop1
        jmp     loop1

loopend1:

...
```

Example – sumarray.c

- See ‘sumarry.c’ in your handouts
- What it does
  - Reads up to 100 integers
  - Computes and writes their sum
- How it works
  - Use an array
  - Reads the integer into an array (loop1)
  - Traverse the array to compute the sum (loop2)

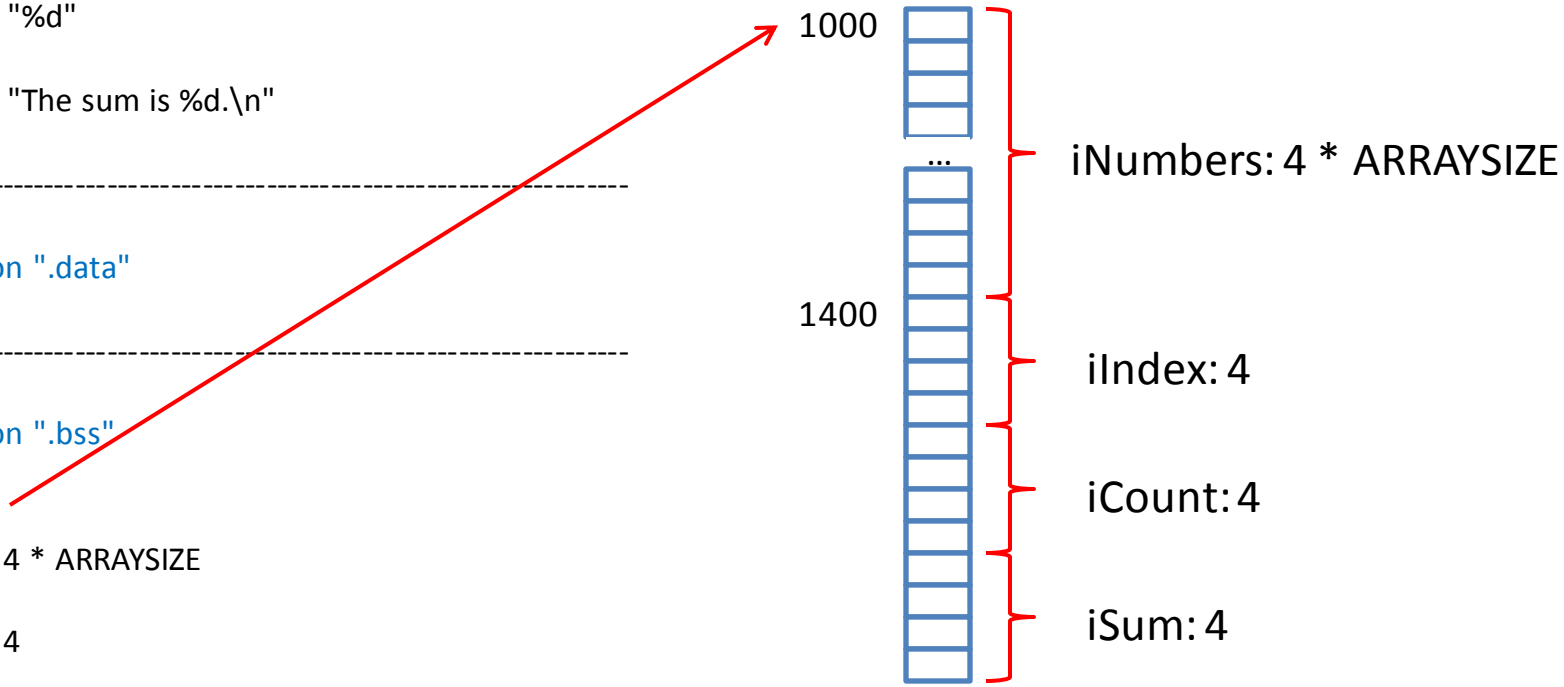
```c
#include <stdio.h>

enum {ARRAYSIZE = 100};

static int aiNumbers[ARRAYSIZE];
static int iIndex;
static int iCount;
static int iSum;

int main(void)

/* Read up to ARRAYSIZE integers from stdin, and write to stdout the
   sum of those integers.  Return 0. */

{
  printf("How many integers?  ");
  scanf("%d", &iCount);

  for (iIndex = 0; iIndex < iCount; iIndex++)
    scanf("%d", &aiNumbers[iIndex]);

  iSum = 0;
  for (iIndex = 0; iIndex < iCount; iIndex++)
    iSum += aiNumbers[iIndex];

  printf("The sum is %d.\n", iSum);

  return 0;
}
```

Loop1

Loop2

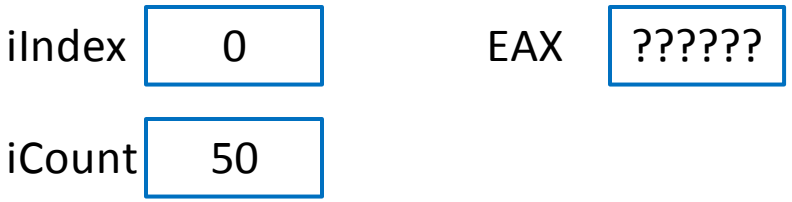
- See ‘sumarryindirect.s’
- How it works
  - Use indirect addressing
  - ex) (%eax)

```
        .equ    ARRAYSIZE, 100

### --------------------------------------------------------------

        .section ".rodata"

cPrompt:
        .asciz  "How many integers?  "
cScanfFormat:
        .asciz  "%d"
cResult:
        .asciz  "The sum is %d.\n"

### --------------------------------------------------------------

        .section ".data"

### --------------------------------------------------------------

        .section ".bss"

iNumbers:
        .skip   4 * ARRAYSIZE
iIndex:
        .skip   4
iCount:
        .skip   4
iSum:
        .skip   4
```

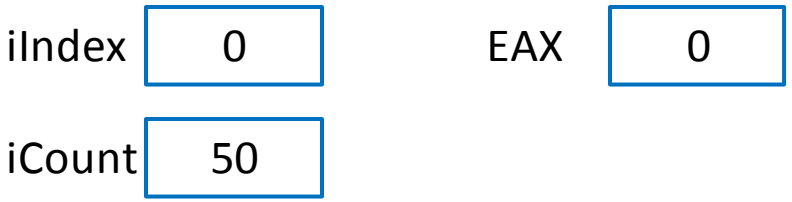


```
### ------------------------------------------------------------

        .section ".text"

        ## ----------------------------------------------------
        ## int main(void)
        ## Read up to ARRAYSIZE integers from stdin, and write to
        ## stdout the sum of those integers.  Return 0.
        ## ----------------------------------------------------

        .globl  main
        .type   main,@function

main:

        pushl   %ebp
        movl    %esp, %ebp

        ## iIndex = 0
        movl    $0, iIndex

        ## printf("How many integers?  ")
        pushl   $cPrompt
        call    printf
        addl    $4, %esp

        ## scanf("%d", &iCount)
        pushl   $iCount
        pushl   $cScanfFormat
        call    scanf
        addl    $8, %esp
```

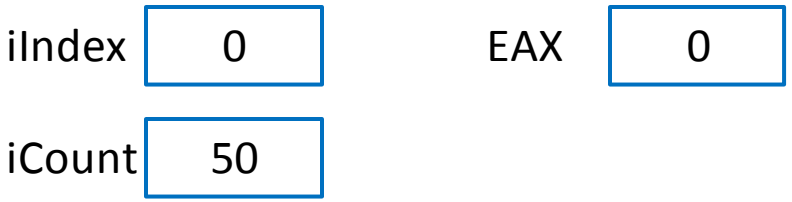
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



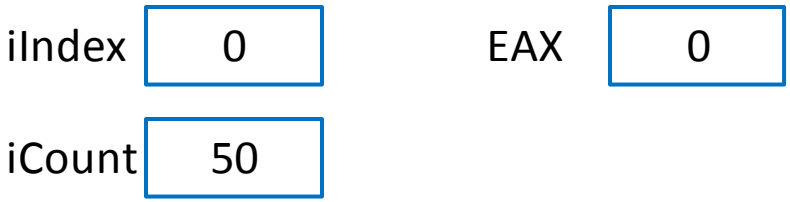
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



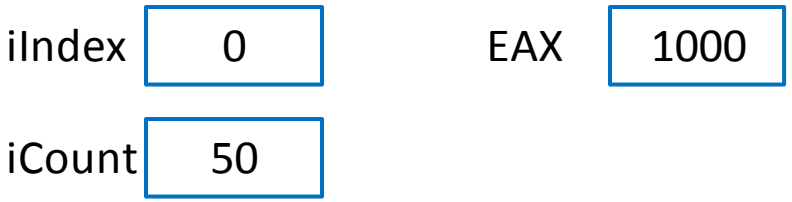
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



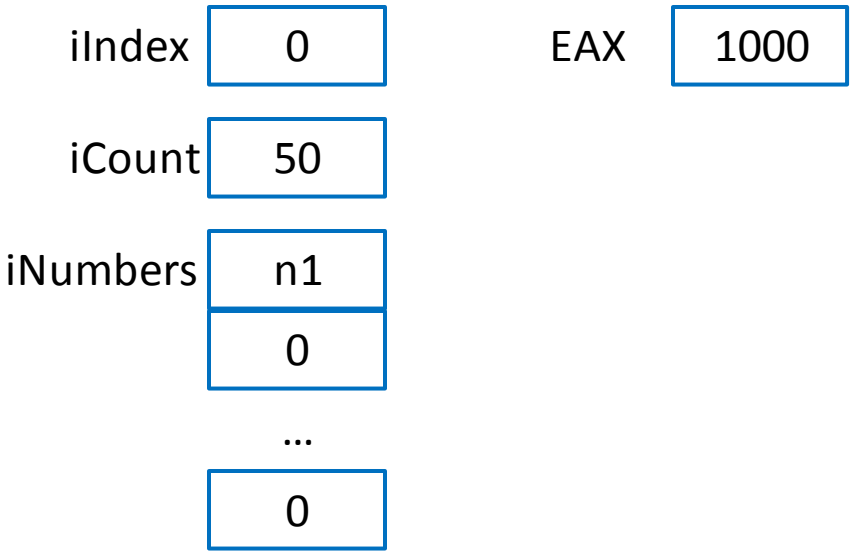
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



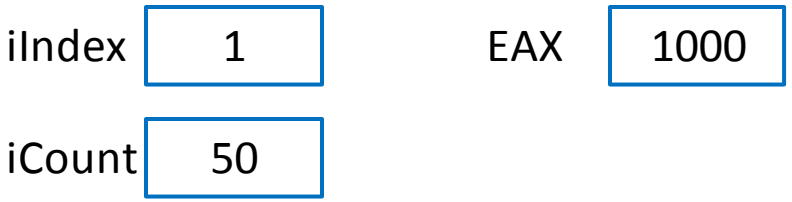
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



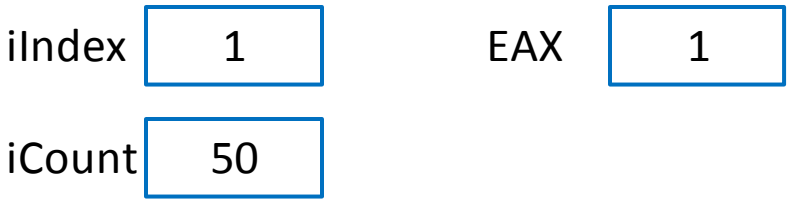
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



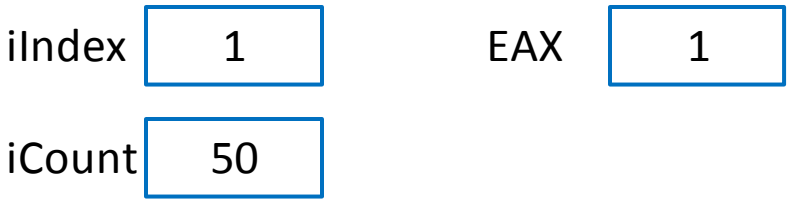
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```

iIndex	1
iCount	50

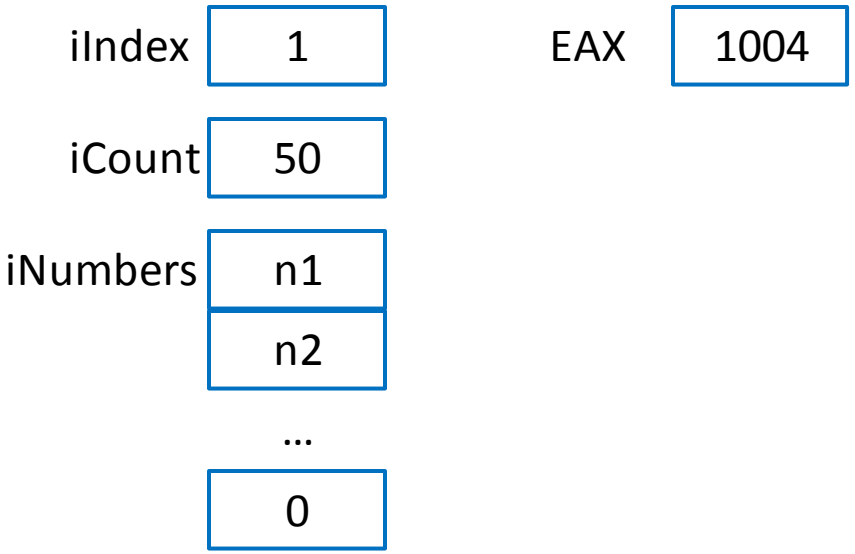
EAX	4

```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```

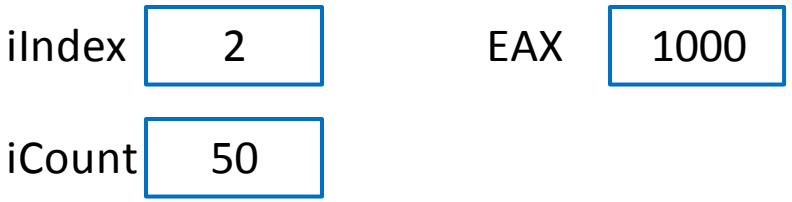
iIndex	1
iCount	50

EAX	1004

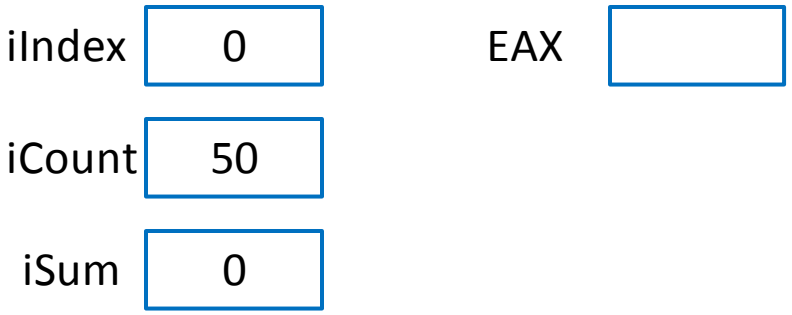
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



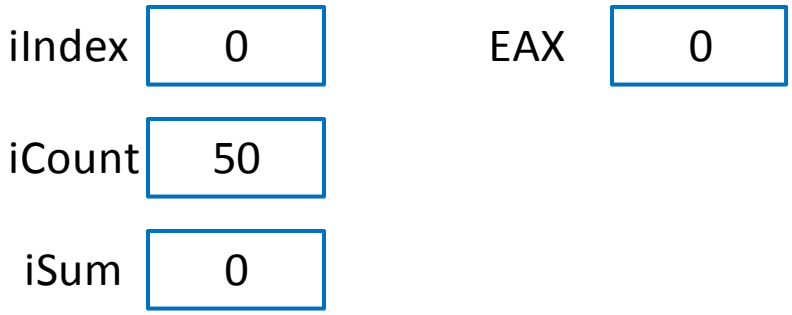
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



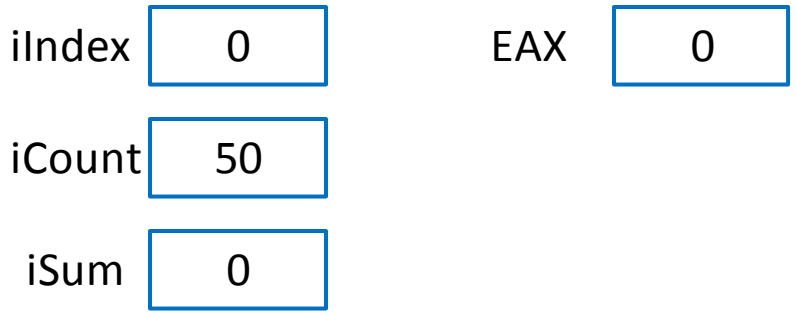
```
loop1:

    ## if (iIndex >= iCount) goto loopend1
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend1

    ## scanf("%d", &piNumbers[iIndex])
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    pushl   %eax
    pushl   $cScanfFormat
    call    scanf
    addl    $8, %esp

    ## iIndex++
    incl    iIndex

    ## goto loop1
    jmp     loop1

loopend1:

    ## iSum = 0
    movl    $0, iSum

    ## iIndex = 0
    movl    $0, iIndex
```



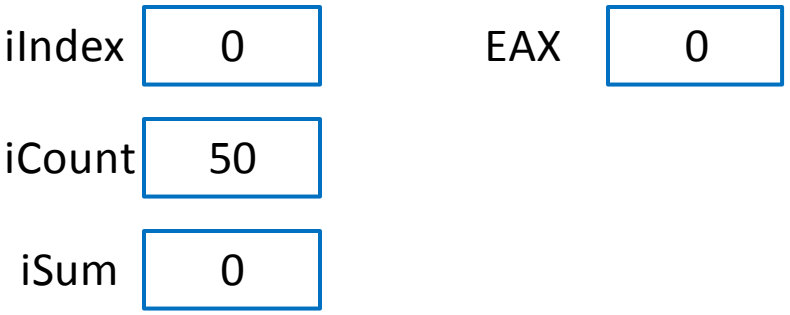
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



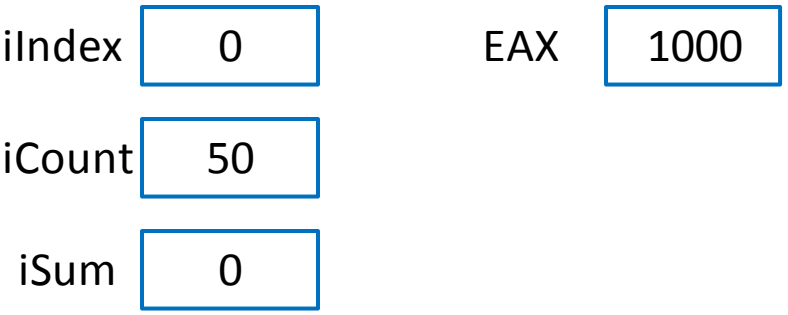
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



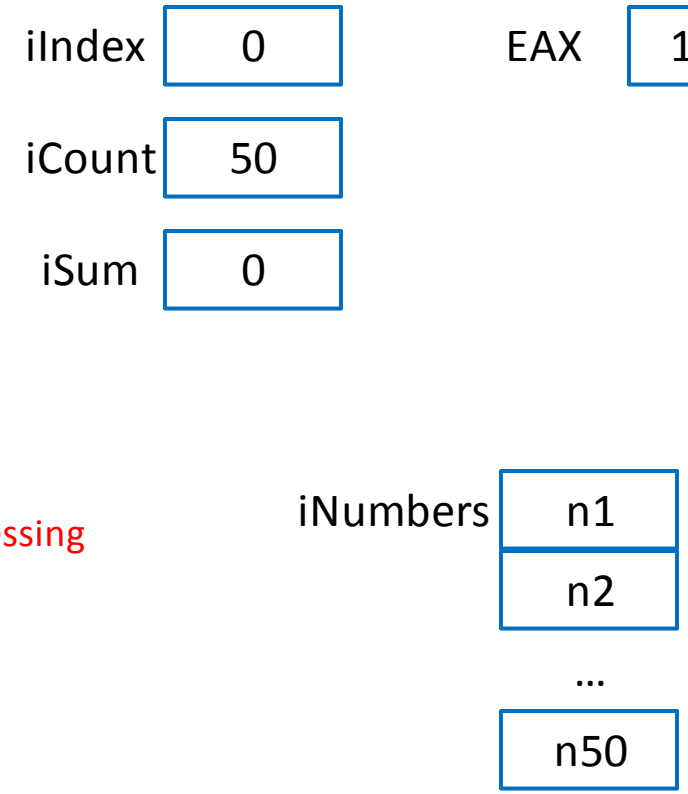
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```

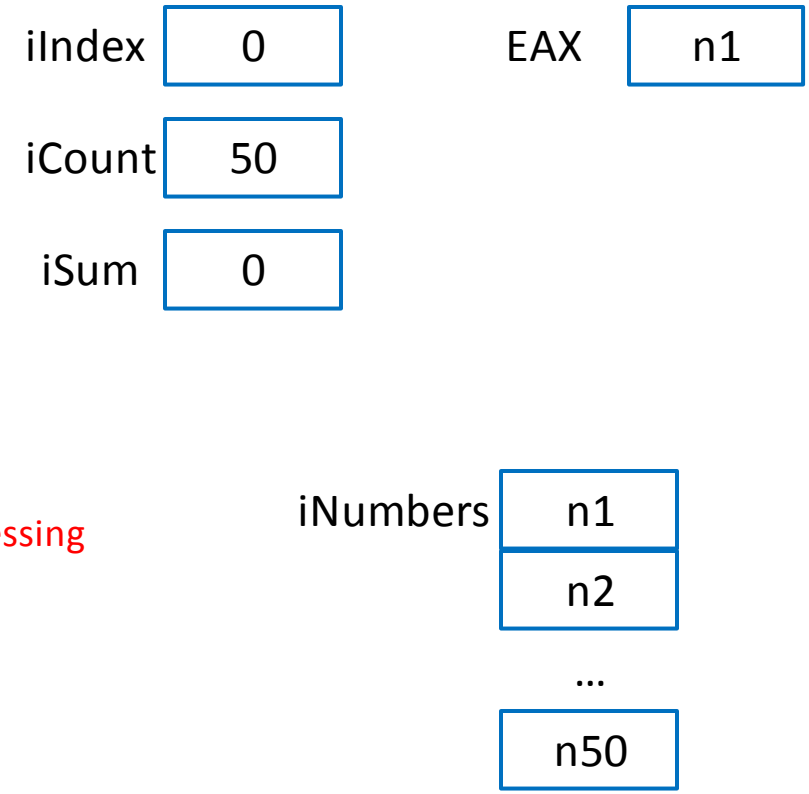


```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



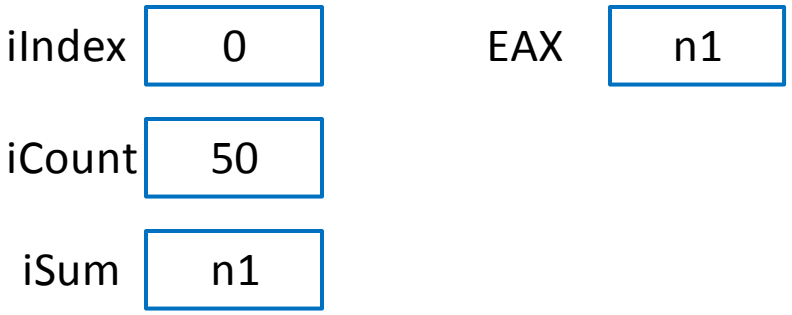
loop2:

```
## if (iIndex >= iCount) goto loopend2
movl    iIndex, %eax
cmpl    iCount, %eax
jge     loopend2

## iSum += piNumbers[iIndex]
movl    iIndex, %eax
sall    $2, %eax
addl    $iNumbers, %eax
movl    (%eax), %eax      # indirect addressing
addl    %eax, iSum

## iIndex++
incl    iIndex

## goto loop2
jmp     loop2
```

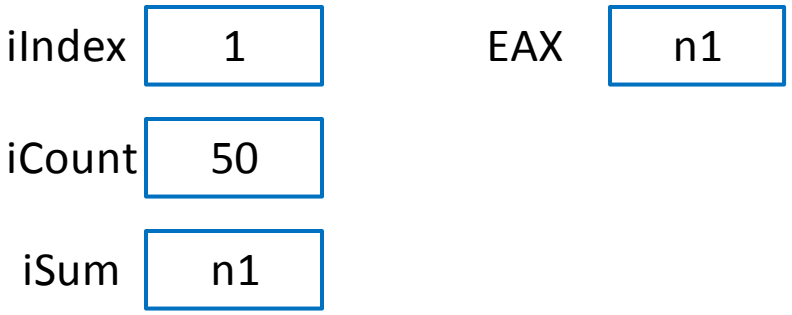


loopend2:

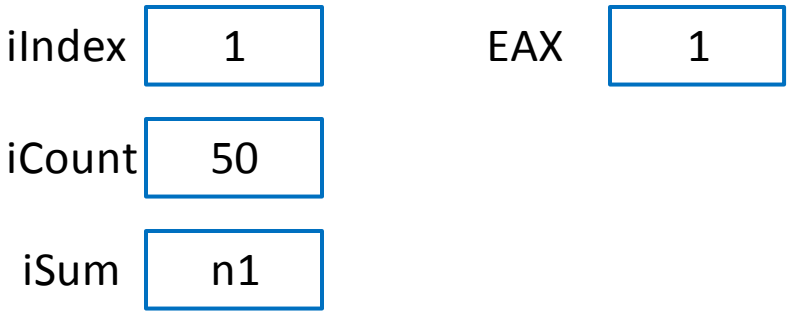
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



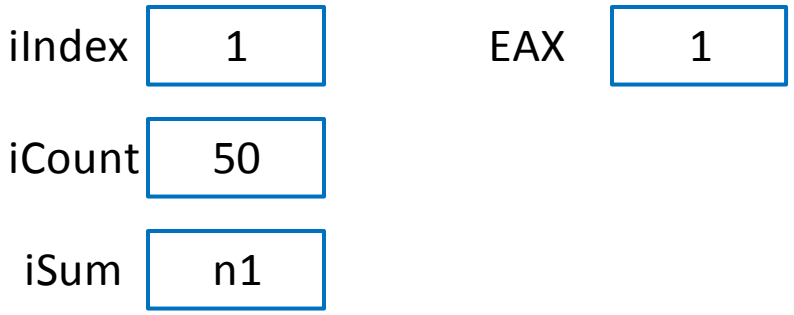
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



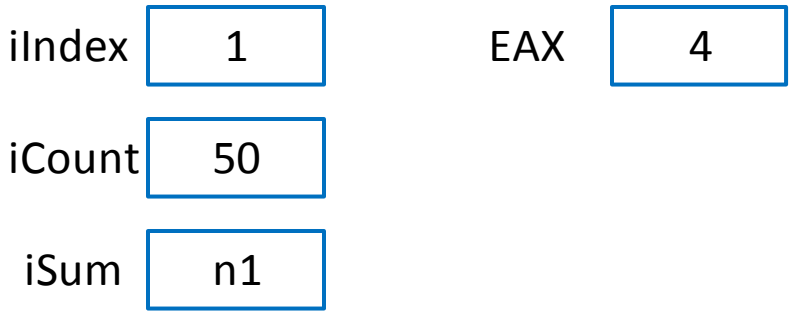
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



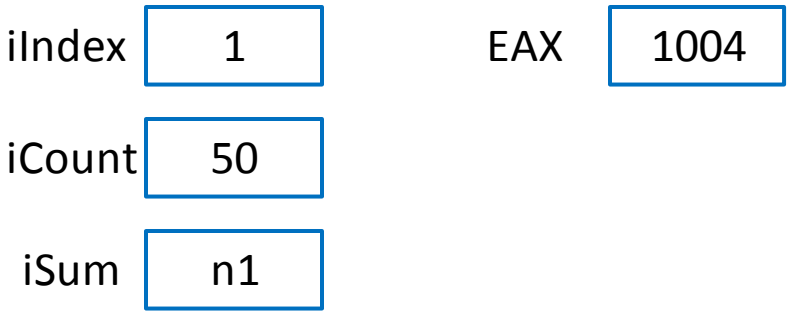
```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```

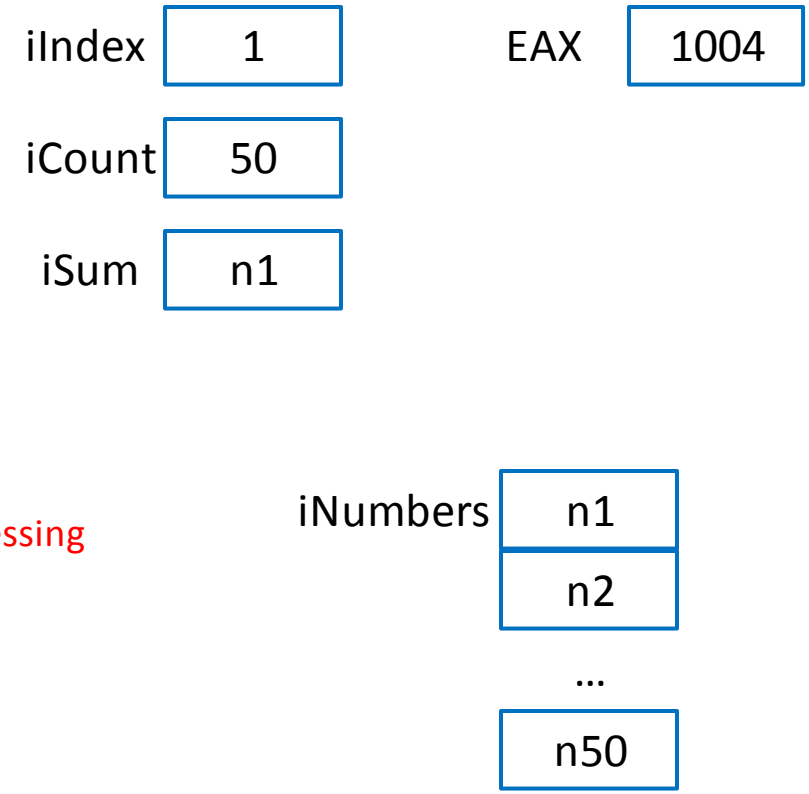


```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



loop2:

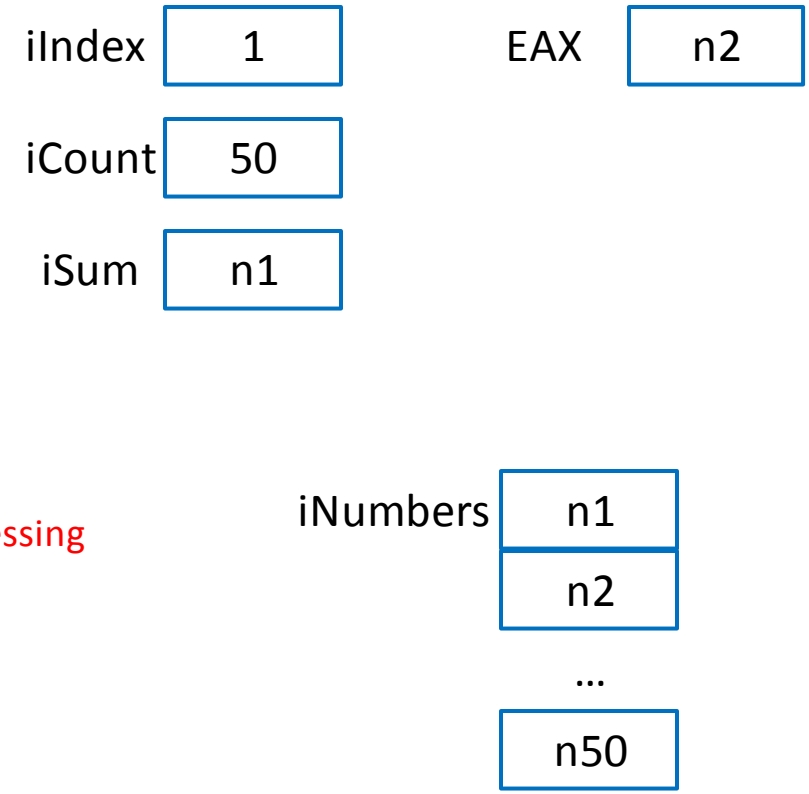
```
## if (iIndex >= iCount) goto loopend2
movl    iIndex, %eax
cmpl    iCount, %eax
jge     loopend2

## iSum += piNumbers[iIndex]
movl    iIndex, %eax
sall    $2, %eax
addl    $iNumbers, %eax
movl    (%eax), %eax      # indirect addressing
addl    %eax, iSum

## iIndex++
incl    iIndex

## goto loop2
jmp     loop2
```



loopend2:

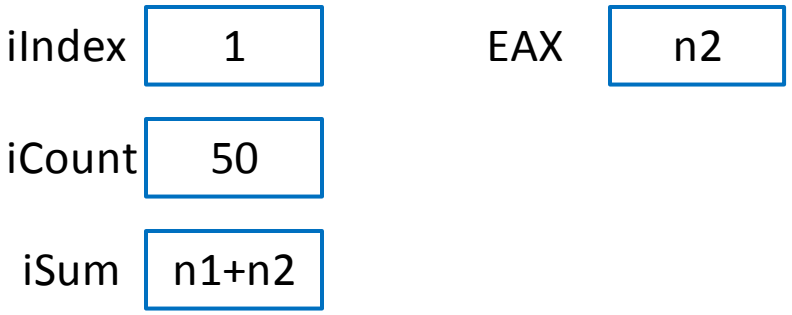
loop2:

```
## if (iIndex >= iCount) goto loopend2
movl    iIndex, %eax
cmpl    iCount, %eax
jge     loopend2

## iSum += piNumbers[iIndex]
movl    iIndex, %eax
sall    $2, %eax
addl    $iNumbers, %eax
movl    (%eax), %eax      # indirect addressing
addl    %eax, iSum

## iIndex++
incl    iIndex

## goto loop2
jmp     loop2
```

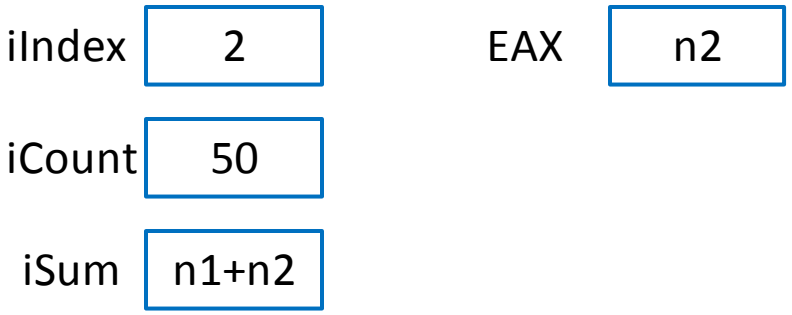


loopend2:

```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



```
loop2:

    ## if (iIndex >= iCount) goto loopend2
    movl    iIndex, %eax
    cmpl    iCount, %eax
    jge     loopend2

    ## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    addl    $iNumbers, %eax
    movl    (%eax), %eax      # indirect addressing
    addl    %eax, iSum

    ## iIndex++
    incl    iIndex

    ## goto loop2
    jmp     loop2

loopend2:
```



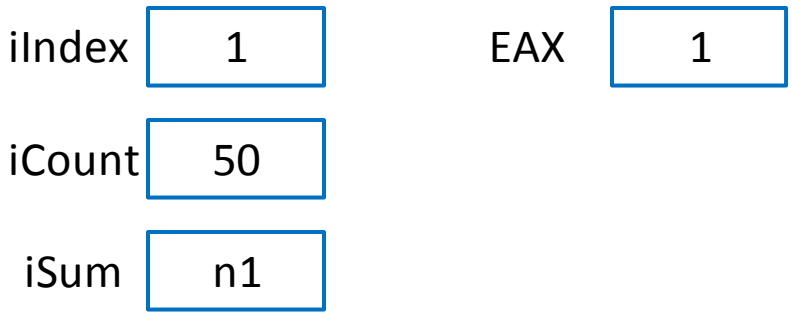
```
loop2:

...

        ## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        addl    $iNumbers, %eax
        movl    (%eax), %eax      # indirect addressing
        addl    %eax, iSum

...

loopend2:
```

Example: sumarrayindexed1.s

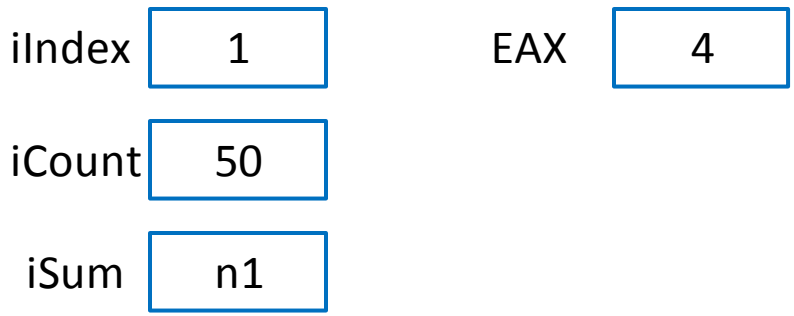
- What it does
  - The same!!
- The difference
  - Use indexed addressing
    - ex) iNumbers(%eax)
    - semantic: mem[iNumbers + reg[EAX]]

```
loop2:

…

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        movl    iNumbers(%eax), %eax      # indexed addressing
        addl    %eax, iSum
…

loopend2:
```

```
loop2:

...

## iSum += piNumbers[iIndex]
    movl    iIndex, %eax
    sall    $2, %eax
    movl    iNumbers(%eax), %eax     # indexed addressing
    addl    %eax, iSum
...

loopend2:
```



```
loop2:

...

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        movl    iNumbers(%eax), %eax     # indexed addressing
        addl    %eax, iSum
...

loopend2:
```



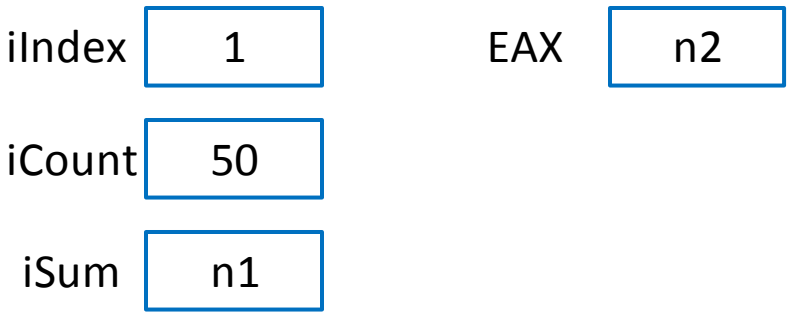
```
loop2:

...

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        movl    iNumbers(%eax), %eax    # indexed addressing
        addl    %eax, iSum
...

loopend2:
```

iIndex	1
iCount	50
iSum	n1

EAX	4

iNumbers(%eax) == mem[iNumbers + reg[EAX]]

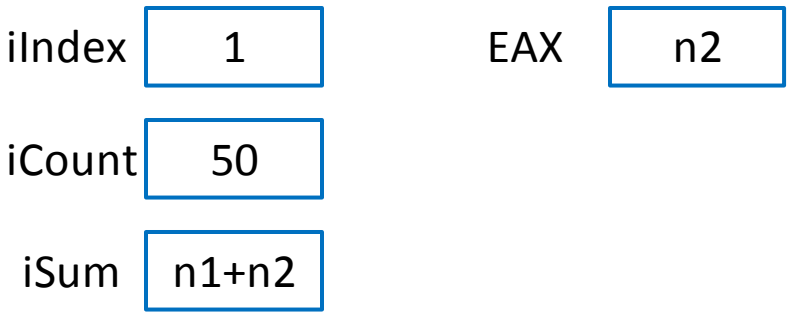
```
loop2:

…

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        movl    iNumbers(%eax), %eax    # indexed addressing
        addl    %eax, iSum
…

loopend2:
```



iNumbers(%eax) == mem[iNumbers + reg[EAX]]

iNumbers(%eax) == mem[1004]

```
loop2:

...

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        movl    iNumbers(%eax), %eax    # indexed addressing
        addl    %eax, iSum
...

loopend2:
```



```
loop2:

...

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        sall    $2, %eax
        movl    iNumbers(%eax), %eax     # indexed addressing
        addl    %eax, iSum
...

loopend2:
```

```
addl    $iNumbers, %eax
movl    (%eax), %eax     # indirect addressing
```

Example: sumarrayindexed2.s

- What it does
  - The same!!
- The difference
  - Use fancy indexed addressing
    - ex) iNumbers(,%ecx, 4)
    - semantic: mem[iNumbers + reg[ECX]*4]

```
loop2:

…

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        movl    iNumbers(,%eax,4), %eax          # fancy indexed addressing
        addl    %eax, iSum

…

loopend2:
```

```
loop2:

...

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        movl    iNumbers(,%eax,4),  %eax        # fancy indexed addressing
        addl    %eax, iSum

...

loopend2:
```

```
sall    $2, %eax
movl    iNumbers(%eax), %eax      # indexed addressing
```

```
loop2:

...

## iSum += piNumbers[iIndex]
        movl    iIndex, %eax
        movl    iNumbers(,%eax,4),  %eax        # fancy indexed addressing
        addl    %eax, iSum

...

loopend2:
```

```
sall    $2, %eax
movl    iNumbers(%eax), %eax      # indexed addressing
```

```
sall    $2, %eax
addl    $iNumbers, %eax
movl    (%eax), %eax      # indirect addressing
```

GDB Debugger

- Please refer to the handout
- New GDB commands (power.s):
  - `info registers` (prints all register values)
  - `print/d $eax`(prints individual register value. Note '$')
  - `print/d iBase` (prints iBase value in decimal)
  - `print/c cPrompt1` (prints 1st byte of mem starting @ cPrompt1)
  - `x/d &iBase` (prints iBase address and dereferenced value)
  - `x/c &cPrompt1`
  - `x/s &cPrompt1`