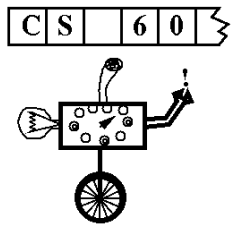


ISC Assembly Language

December 5, 2001

Computer Architecture

- We will work with a simulated tutorial architecture:

  ISC: *Incredibly Simple Computer*

- ISC is an example of a

  RISC (**R**educed **I**nstruction **S**et **C**omputer)**,**

  as opposed to a

  CISC (**C**omplex **I**nstruction **S**et **C**omputer)

Instructions

- *Machine language* is the language understood directly by a computer chip.
- A machine language program is a sequence of *instructions* stored in memory.
- Each instruction is a 32-bit word that tells the ISC to perform a single (simple) operation
  - Add the values in two registers
  - Get the value in a memory address
  - Start executing the program at a different memory address
  - etc.

Registers

- The ISC hardware contains 32 *general-purpose registers* that instructions can use for computations.
  - Each register can hold a 32-bit number
  - Registers are numbered 0-31
- Other registers exist, but are used implicitly
  - The most important is the instruction pointer (IP), which always contains the address of the next instruction to execute.
  - In some computers the instruction pointer is called the program counter (PC)

Machine-Language vs. Assembly Langauge

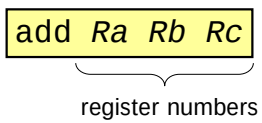
- Programming of the bare machine is typically done in *assembly language*
  - Really don't want to write a program as a list of 32-bit numbers...
- Languages correspond very directly
  - One line of assembly language is roughly equal to one machine instruction
  - However, assembly language is (more) human-readable
  - Translation from assembly to machine language is done by a program called an *assembler*.
  - The assembler may also permit the programmer to use abbreviations (particularly by giving *names* that represent registers or memory addresses)

Programming in Assembly Language

- Programming in assembly language reminds one of the Japanese saying about climbing Mt. Fuji:

> To never have climbed Mt. Fuji is to be a fool.
> Only a fool would climb Mt. Fuji more than once.

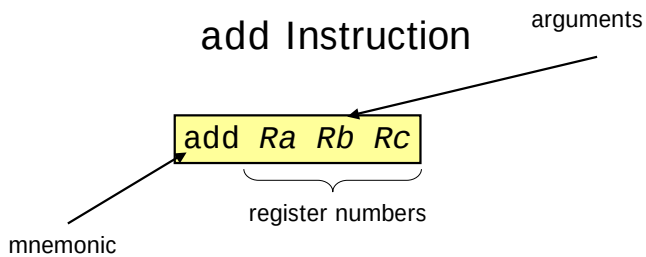
add Instruction



Example: `add 2 3 16`

Effect: `reg[Ra] = reg[Rb]+reg[Rc]`

add Instruction



Example: `add 2 3 16`

Effect: `reg[Ra] = reg[Rb] + reg[Rc]`

(Bitwise) `or` Instruction

`or Ra Rb Rc`

Example: `or 9 2 1`

Effect: `reg[Ra] = reg[Rb] | reg[Rc]`

Other Register Instructions

- Many other arithmetic and logical instructions follow a similar pattern

```
sub Ra Rb Rc
mul Ra Rb Rc
div Ra Rb Rc
and Ra Rb Rc
shr Ra Rb Rc
shl Ra Rb Rc
```

shift right

shift left

Other Register Instructions

- Some register instructions only need two arguments…

`comp Ra Rb`

bitwise not

Effect: `reg[Ra] = ~reg[Rb]`

`copy Ra Rb`

Effect: `reg[Ra] = reg[Rb]`

Other Register Instructions

- …or even just one register argument

up to 24-bits

`lim Ra Constant`

Example: `lim 13 42`
Effect: `reg[Ra] = Constant`

`aim Ra Constant`

Example: `aim 13 42`
Effect: `reg[Ra] = reg[Ra]+Constant`

Memory Access Instructions

- Memory acts like a big array (of 32-bit words)
  - *Word-addressed*, rather than *byte-addressed*

`load Ra Rb`

Effect: `reg[Ra] = mem[reg[Rb]]`

`store Ra Rb`

Effect: `mem[reg[Ra]] = reg[Rb]`

Jumps

- Recall that the each instruction is stored as a 32-bit word in memory.
- Usually, we execute each instruction and then go on the the next word.
  - The IP always contains the address of the next instruction to execute
  - Normally, IP is 1 more than the address of the current instruction.
- Jumps (and, on other machines, branches) let us change the instruction pointer.

(Unconditional) Jump

`junc Ra`

Example: `junc 2`

Effect: `IP = reg[Ra]`

Conditional Jumps

- Change the IP if two arguments are in a particular relation (less-than, equal, etc.)

`jeq Ra Rb Rc`

Effect:
```
if (reg[Rb]==reg[Rc])
   IP = reg[Ra]
```

```
jeq Ra Rb Rc
jne Ra Rb Rc
jgt Ra Rb Rc
```

```
jgte Ra Rb Rc
jlt  Ra Rb Rc
jlte Ra Rb Rc
```

Jump to Subroutine

`jsub Ra Rb`

Effect:

```
reg[Rb] = IP;
IP = reg[Ra]
```

Why? The subroutine can get back to the instruction following the `jsub` by jumping to the address in *Rb*!

Example (fragment)

```
...
lim 2 0
lim 3 0
lim 6 20
lim 5 14
jlte 6 1 3
load 4 0
add 2 4 2
aim 0 1
aim 1 -1
junc 5
...
```

ISCAL
ISC Assembly Language

- See http://www.cs.hmc.edu/~keller/isc/
- Free-form input, but generally format line-by-line
- Regular instructions
  - *Ra*, *Rb*, *Rc* are register names
  - *C* is a constant
    - **lim** *Ra C*
    - **add** *Ra Rb Rc*
    - **copy** *Ra Rb*
    - **shl** *Ra Rb*
    - **load** *Ra Rb*
    - etc.

ISC Assembly Language

- *Assembler directives*: not instructions to computer, but rather tell assembler what to do:

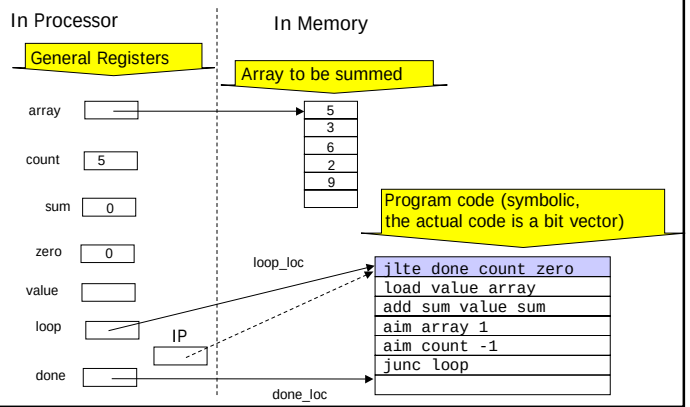
`define Ident Value`	Defines *Ident* to abbreviate *Value*.
`register Ident Reg`	*Ident* can be used for register *Reg*.
`use Identifier`	Defines *Identifier* to name an unused register.
`origin Value`	Put next instruction at specified memory location. The location counter is incremented as loading progresses.
`label Identifier`	Associates current instruction location with *Identifier*.

- See `http://www.cs.hmc.edu/~keller/isc/`

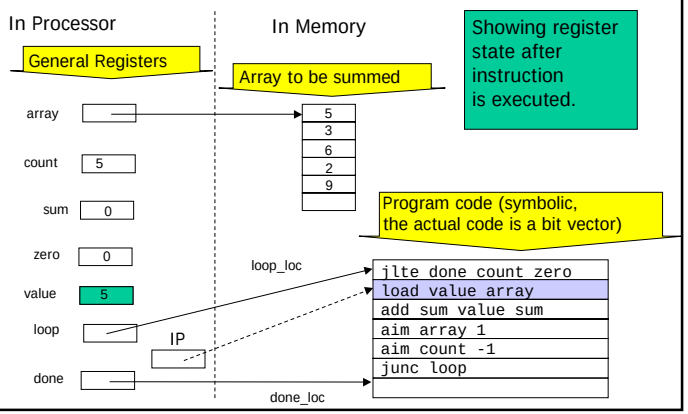
ISCAL code for summing an array

```
    use array  use count             // array base and count
    use sum    use zero   use value  // local registers
    use loop   use done              // address registers

    ... insert code to load array and count values ...

    lim sum  0          // initialize sum
    lim zero  0         // comparison value
    lim done done_loc   // address of instruction following
    lim loop loop_loc   // address of next instruction

label loop_loc

    jlte done count zero // jump if <= 0
    load value array     // load register next array value
    add sum value sum    // add the next number to the sum
    aim array 1          // add 1 to the array address
    aim count -1         // add -1 to the count
    junc loop            // go back and compare

label done_loc
```

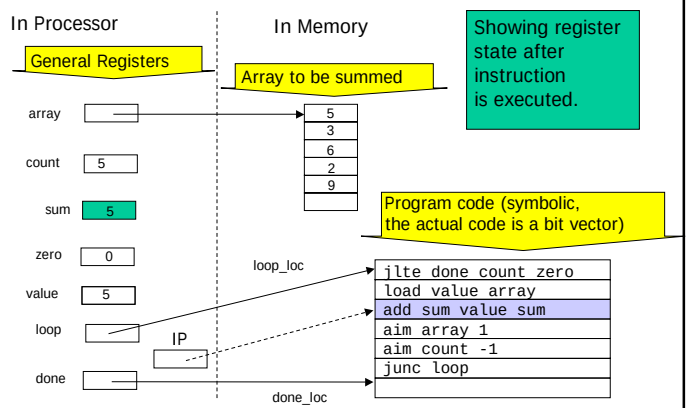
Array Summation

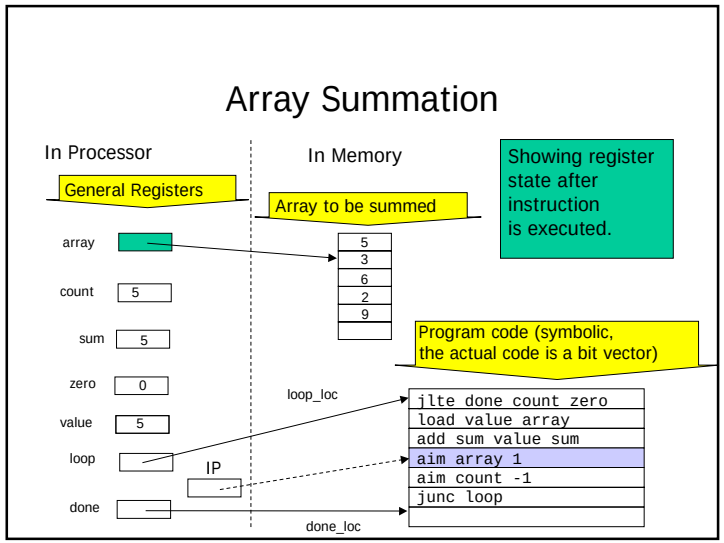


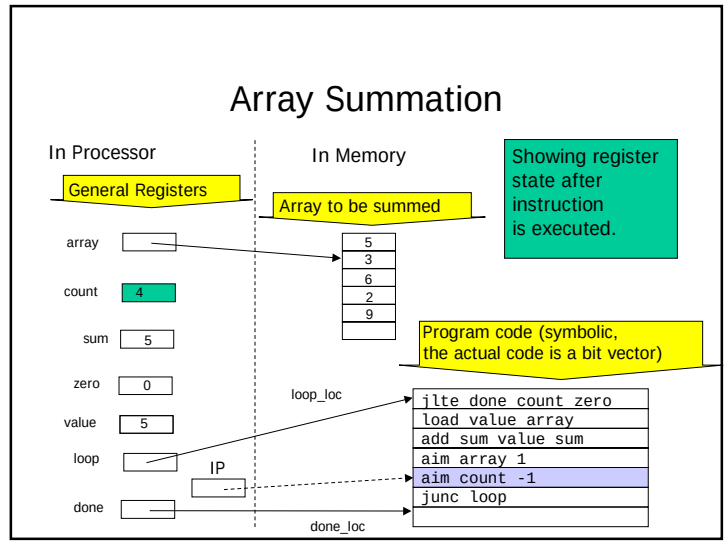
Array Summation

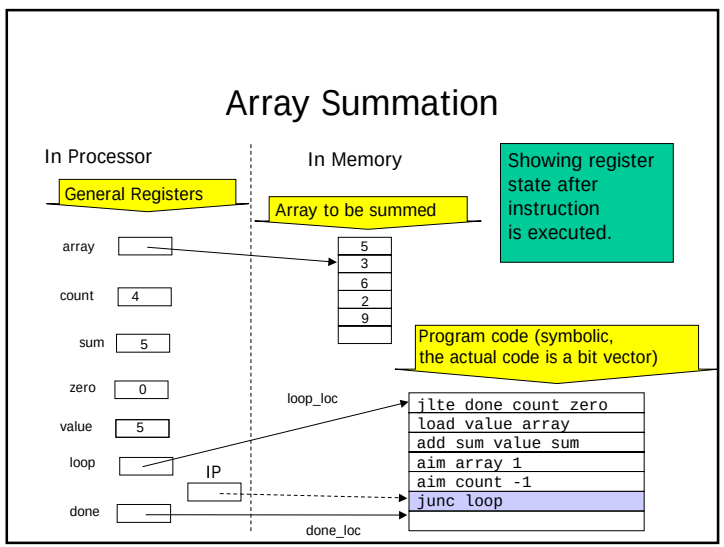


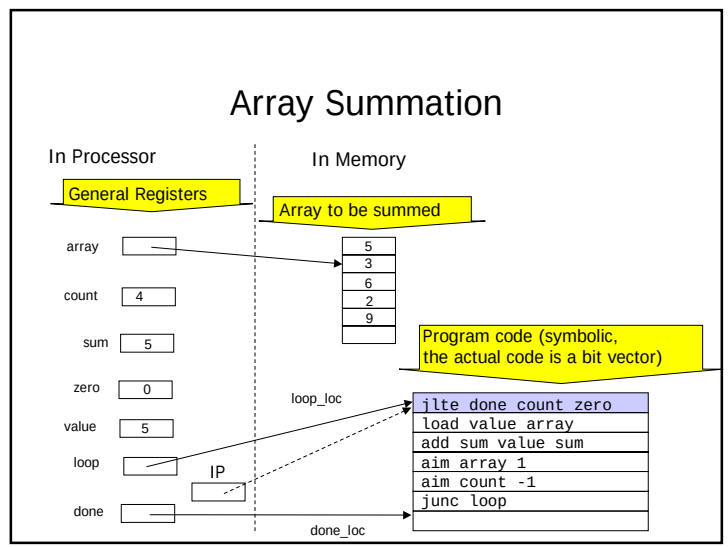
Array Summation

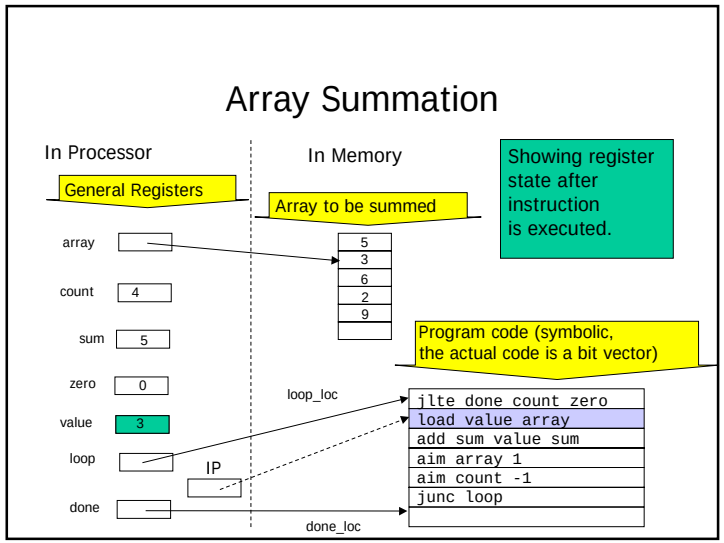


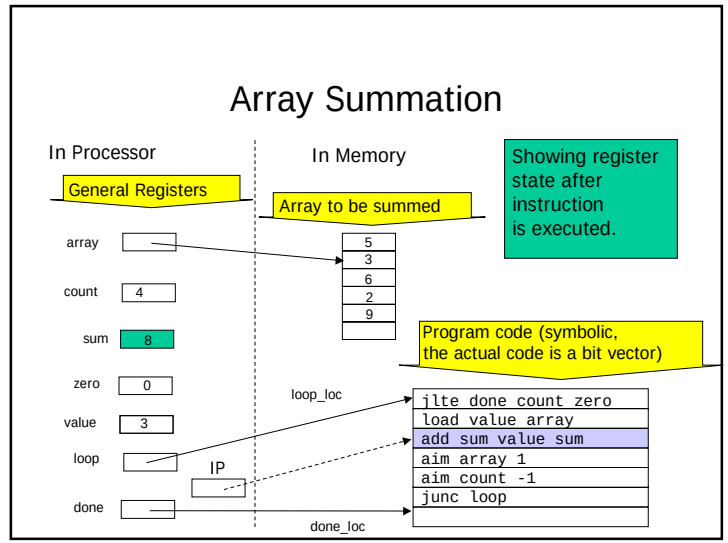

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 5
sum 5
zero 0
value 5
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

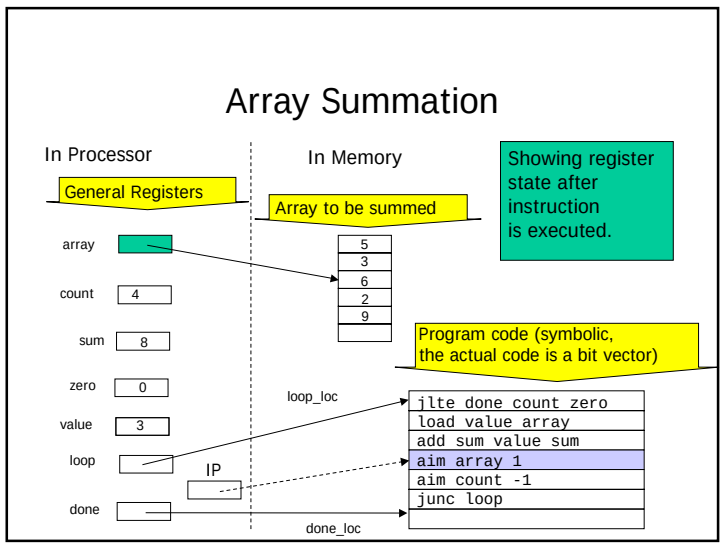

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 4
sum 5
zero 0
value 5
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

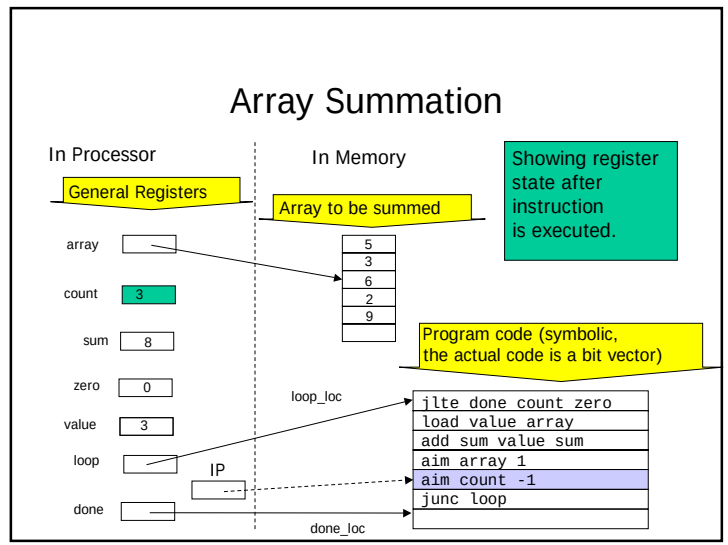

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 4
sum 5
zero 0
value 5
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

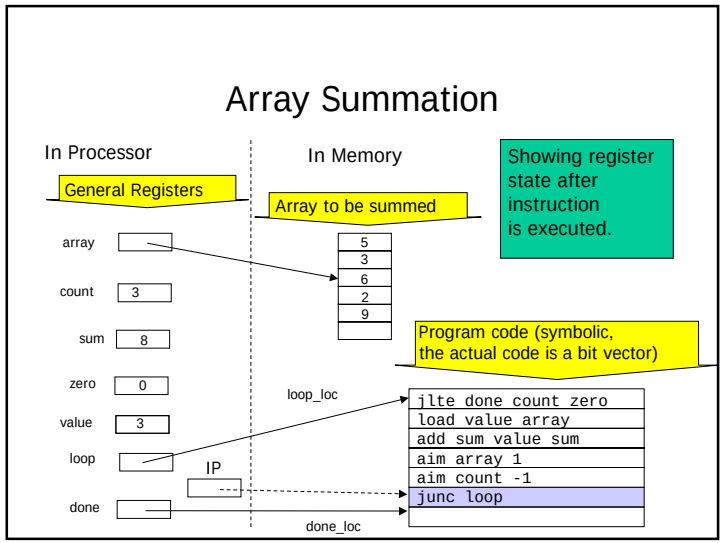

Array Summation
In Processor
In Memory
General Registers
Array to be summed
array
count 4
sum 5
zero 0
value 5
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

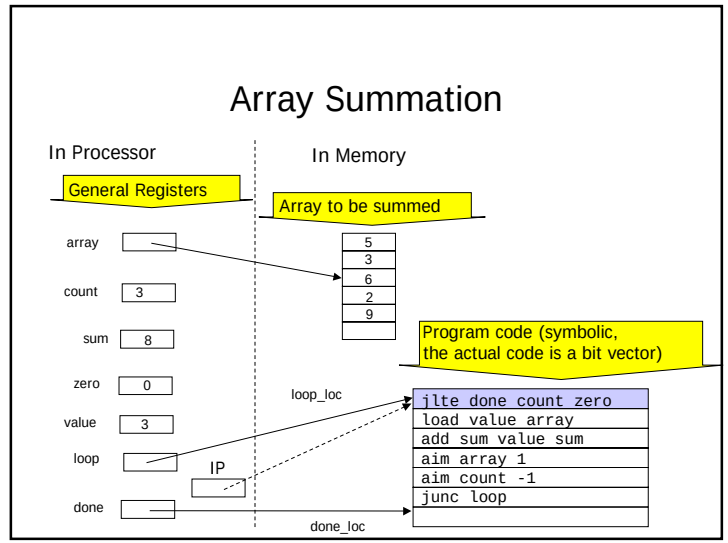

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 4
sum 5
zero 0
value 3
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

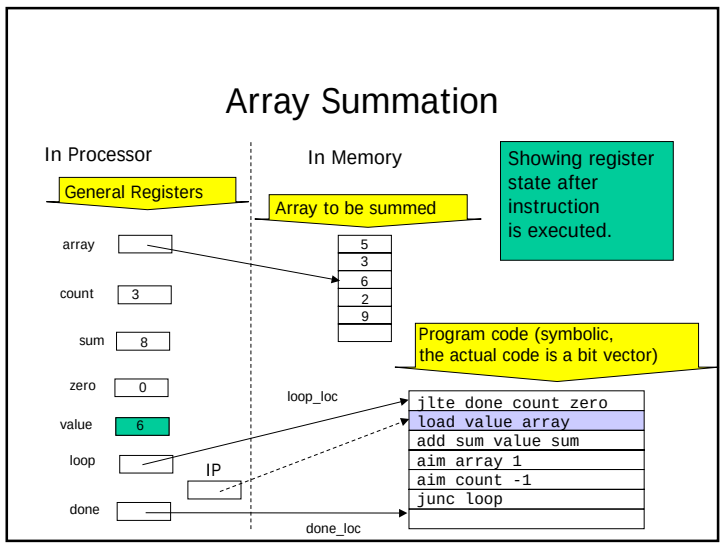

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 4
sum 8
zero 0
value 3
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

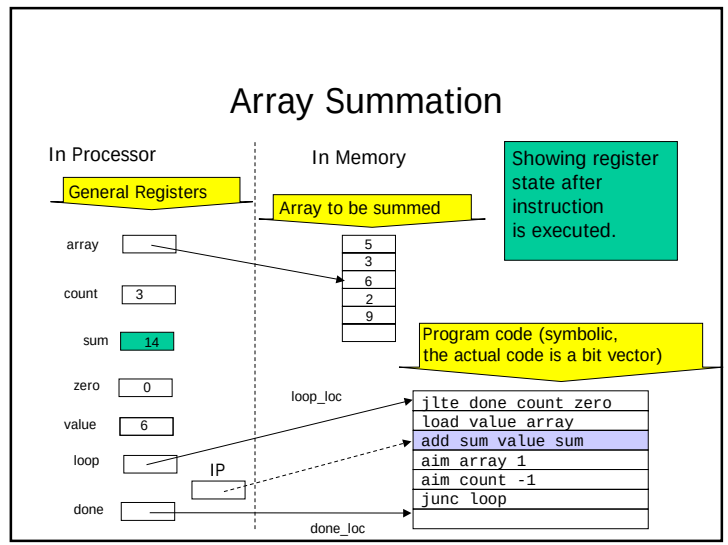

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 4
sum 8
zero 0
value 3
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

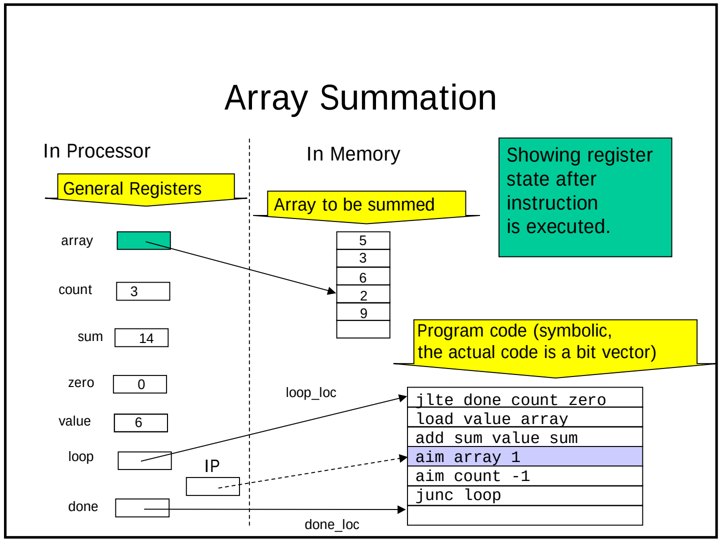

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 3
sum 8
zero 0
value 3
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

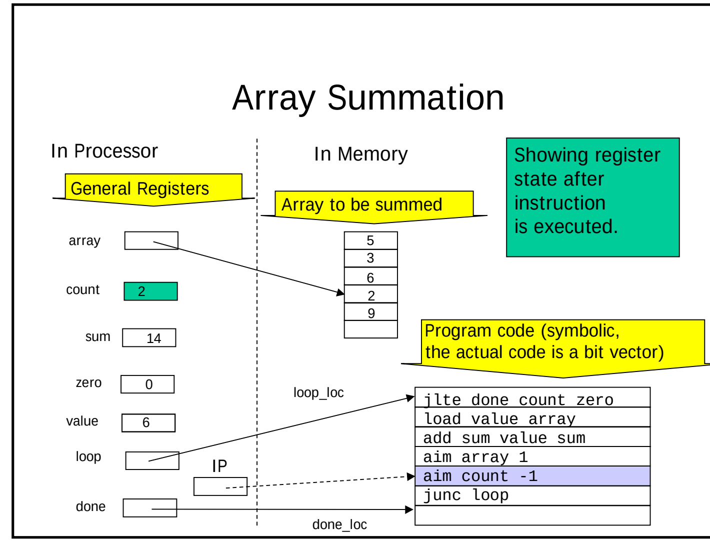

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 3
sum 8
zero 0
value 3
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

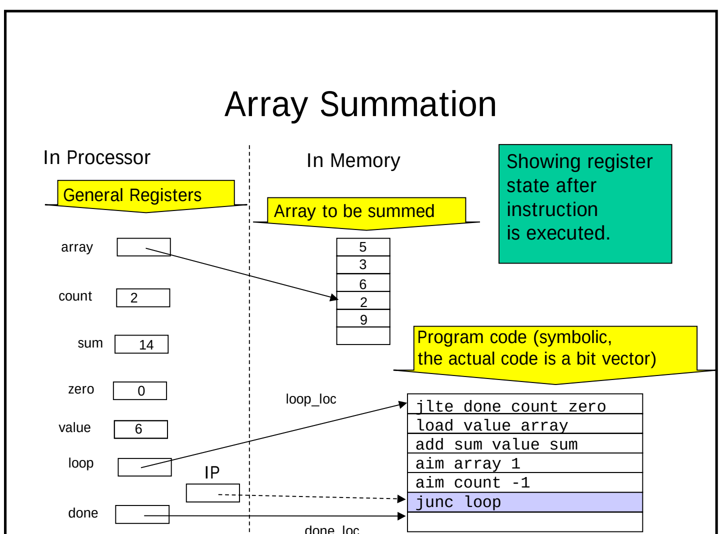

Array Summation
In Processor
In Memory
General Registers
Array to be summed
array
count 3
sum 8
zero 0
value 3
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

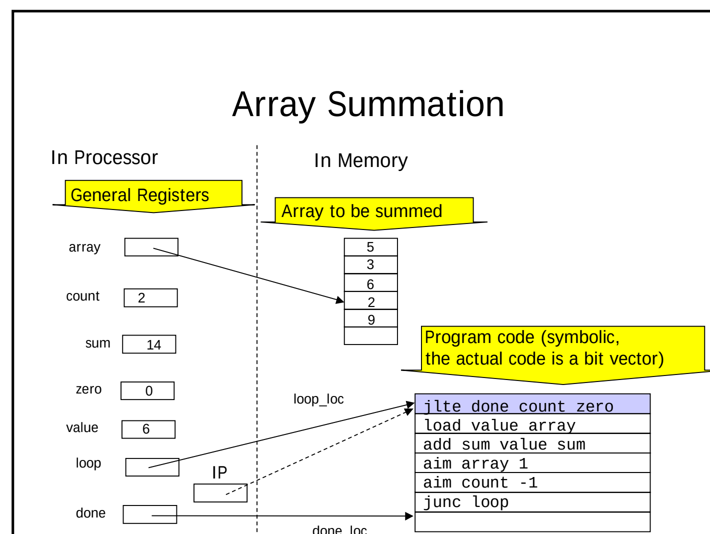

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 3
sum 8
zero 0
value 6
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

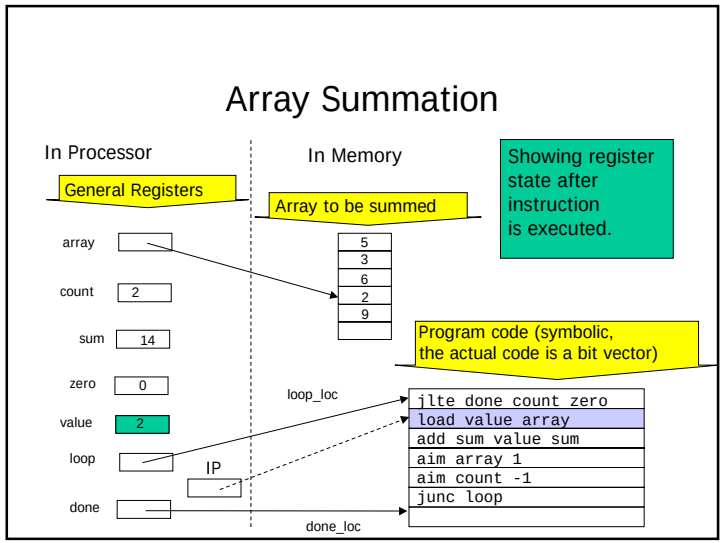

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 3
sum 14
zero 0
value 6
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

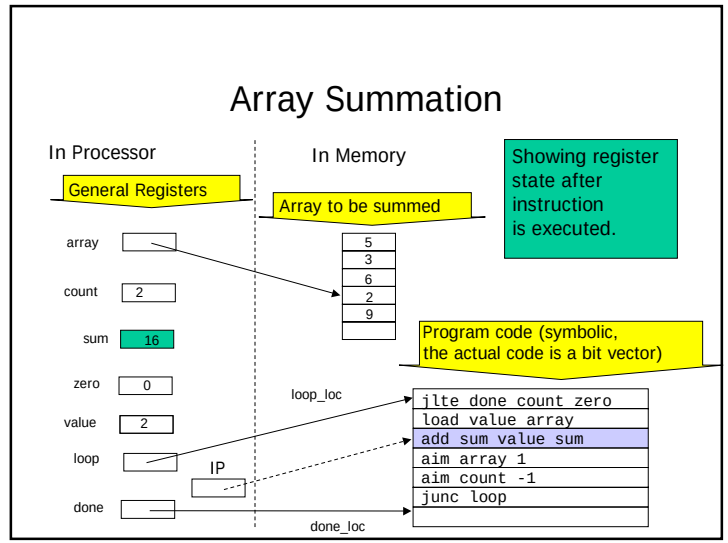

Array Summation
In Processor
In Memory
Showing register state after instruction is executed.
General Registers
Array to be summed
array
count 3
sum 14
zero 0
value 6
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

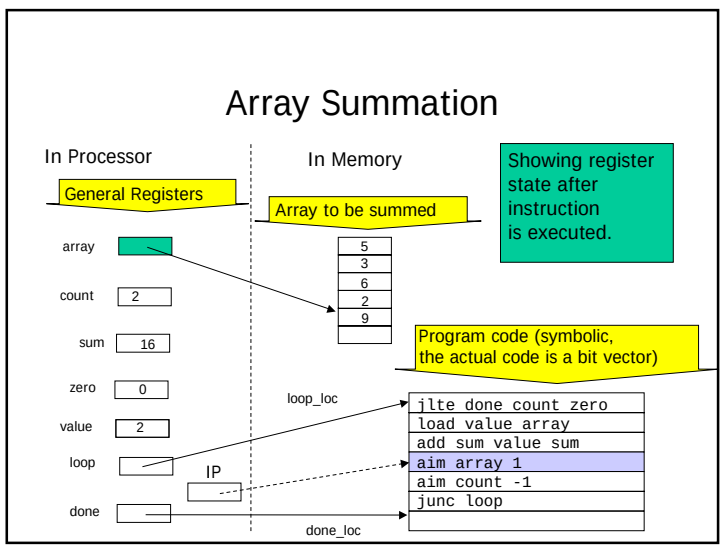

Array Summation
In Processor
In Memory
Showing register state after instruction is executed.
General Registers
Array to be summed
array
count 2
sum 14
zero 0
value 6
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

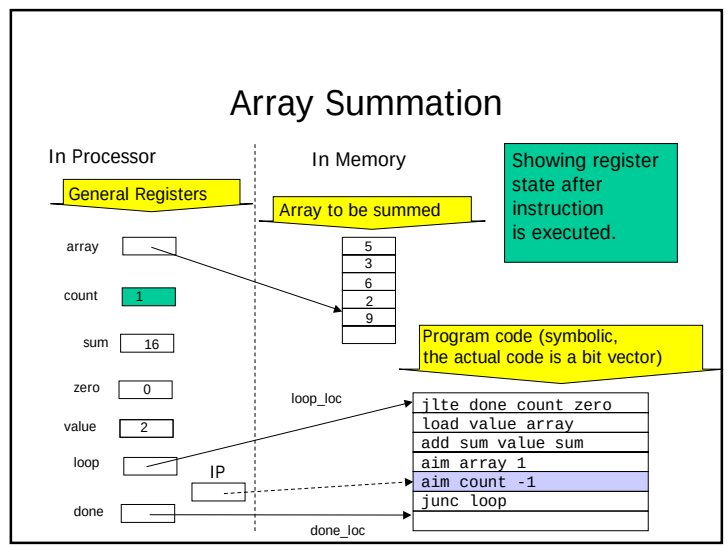

Array Summation
In Processor
In Memory
Showing register state after instruction is executed.
General Registers
Array to be summed
array
count 2
sum 14
zero 0
value 6
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

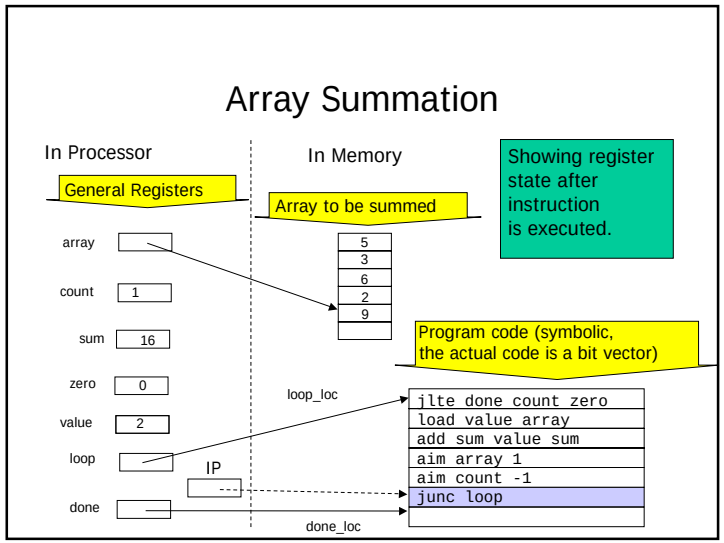

Array Summation
In Processor
In Memory
General Registers
Array to be summed
array
count 2
sum 14
zero 0
value 6
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

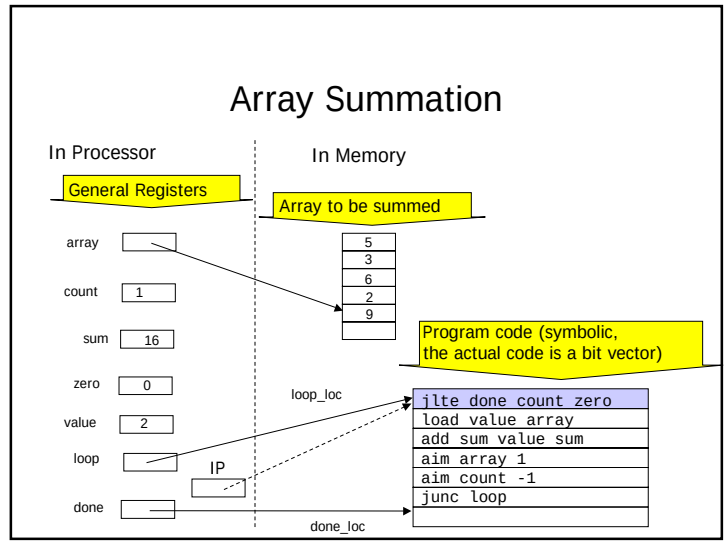

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 2
sum 14
zero 0
value 2
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

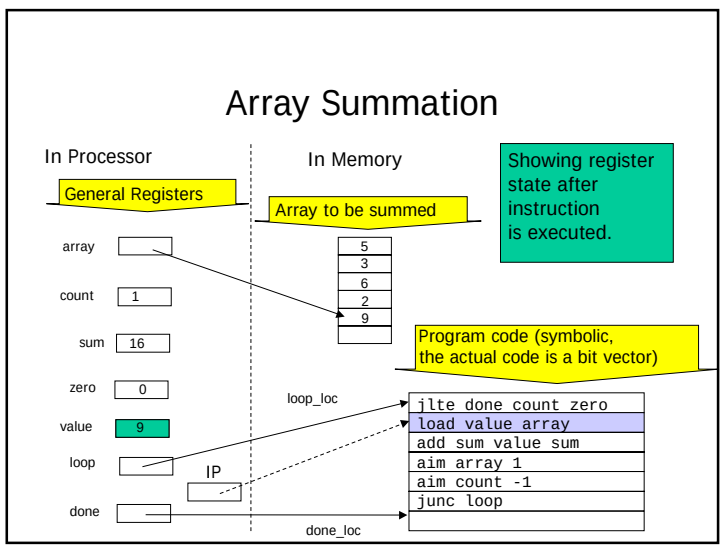

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 2
sum 16
zero 0
value 2
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

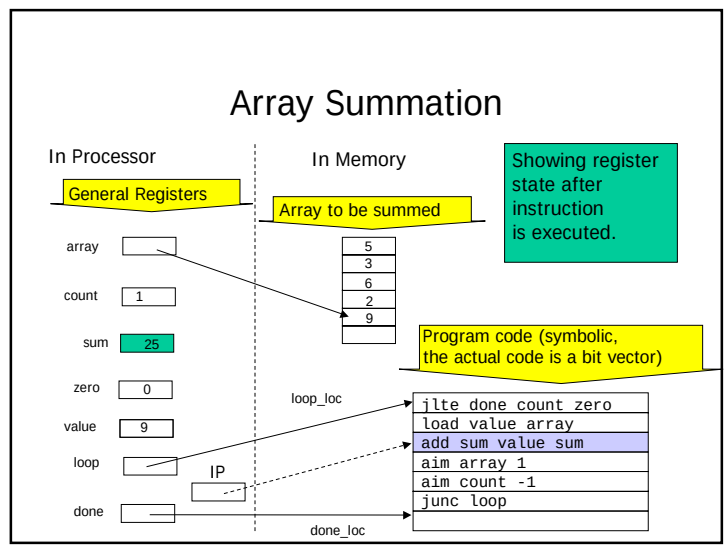

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 2
sum 16
zero 0
value 2
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

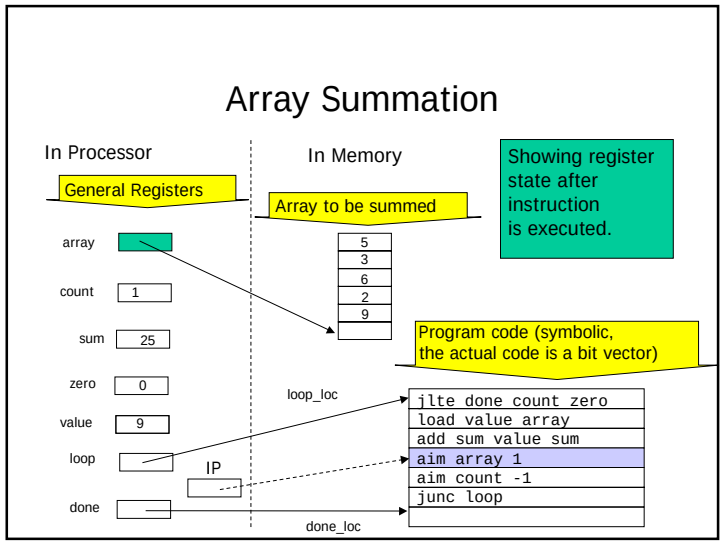

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 1
sum 16
zero 0
value 2
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
done_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop

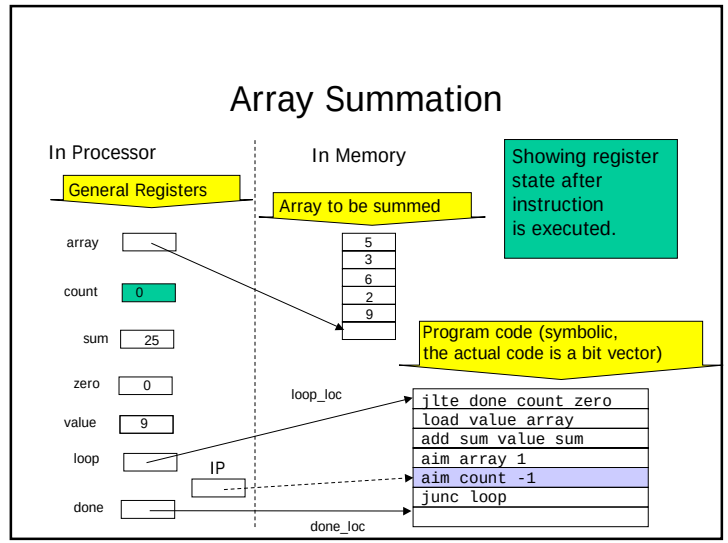

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count
1
sum
16
zero
0
value
2
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

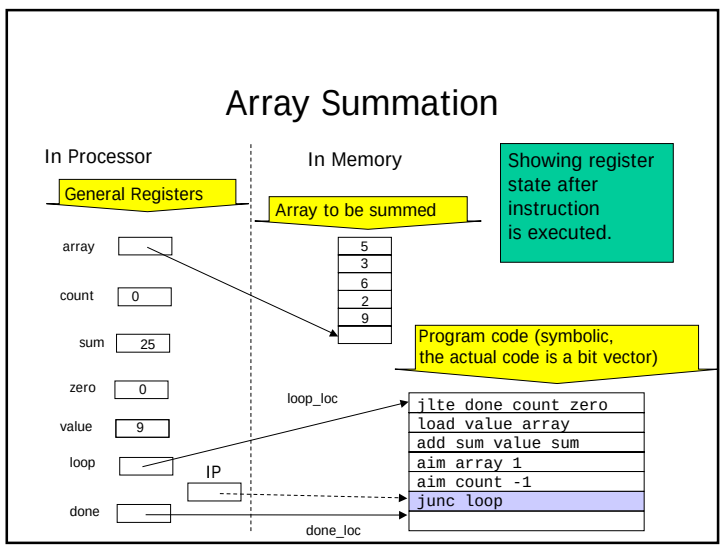

Array Summation
In Processor
In Memory
General Registers
Array to be summed
array
count
1
sum
16
zero
0
value
2
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

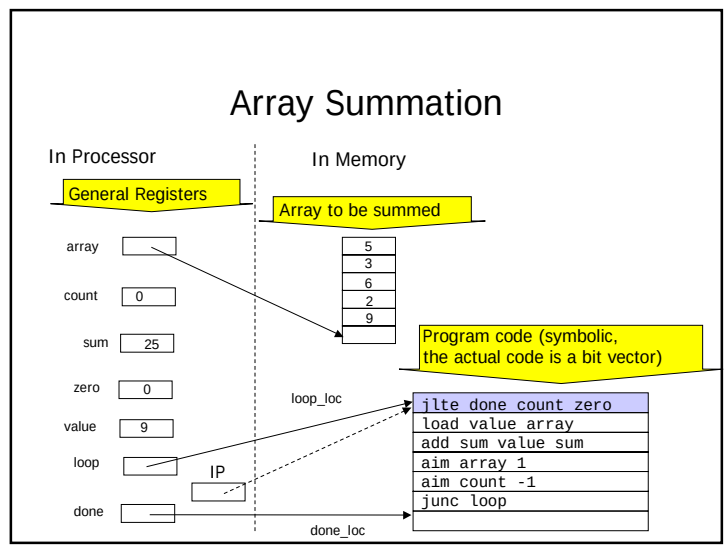

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count
1
sum
16
zero
0
value
9
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc


Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count
1
sum
25
zero
0
value
9
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

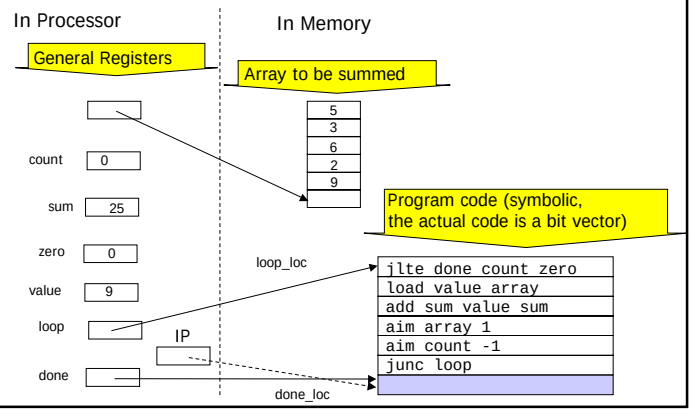

Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 1
sum 25
zero 0
value 9
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc


Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 0
sum 25
zero 0
value 9
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc


Array Summation
In Processor
In Memory
General Registers
Array to be summed
Showing register state after instruction is executed.
array
count 0
sum 25
zero 0
value 9
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc


Array Summation
In Processor
In Memory
General Registers
Array to be summed
array
count 0
sum 25
zero 0
value 9
loop
done
IP
5
3
6
2
9
Program code (symbolic, the actual code is a bit vector)
loop_loc
jlte done count zero
load value array
add sum value sum
aim array 1
aim count -1
junc loop
done_loc

Array Summation



The ISC Assembler

- The ISC assembler is actually a combined assembler, loader, and tracer (for debugging).
  - The executable is `/cs/cs60/bin/isc`
  - Sample programs are in `/cs/cs60/isc/`
- The array summation program is in /cs/cs60/isc/array.isc. It includes additional code to create the array and output the result.