POSIX Threads programming (pthreads)

Concurrent Programming

Introduction

- What is Pthreads?
- Pthreads API
- Example

What is Pthreads?

- Pthreads is a POSIX standard for describing a thread model, it specifies the API and the semantics of the calls.
- Model popular – nowadays practically all major thread libraries on Unix systems are Pthreads-compatible
- Pthreads defines a set of C programming language types, functions and constants.

Pthreads API

- pthread_create
  - More APIs related to pthreads attibute
- pthread_join
- pthread_exit
- pthread_self
- more APIs, but not today

Pthreads API – pthread_create

```
int pthread_create(pthread_t *thread,
                   const pthread_attr_t *attr,
                   void *(*start_routine)(void *),
                   void *arg);
```

- Create a new thread

```
@param[out] thread         ID of a new thread
@param[in]  attr           Used for setting attributes of a new thread.(e.g.,Stack size). Default 0
@param[in]  start_routine  Function to be executed by the new thread
@param[in]  arg            Argument to be passed to the start_routine
@return                    0 if thread creation succeeds
```

Pthreads API – pthread_attr

```
int pthread_attr_init(pthread_attr_t *attr);
int pthread_attr_destroy(pthread_attr_t *attr);
```

- Initialize and destroy thread attributes object

```
@param[in]  attr          Thread attributes object
@return                   0 on success; on error, they return a nonzero error number
```

Pthreads API – pthread_attr_stacksize

```
int pthread_attr_setstacksize(pthread_attr_t *attr,
                              size_t stacksize);
int pthread_attr_getstacksize(pthread_attr_t *attr,
                              size_t* stacksize);
```

- Initialize and destroy thread attributes object

@param[in]	attr	Thread attributes object
@param[in, out]	stacksize	stack size
@return		0 on success; on error, they return a nonzero error number

Pthreads API – pthread_join

```
int pthread_join(pthread_t thread,
                 void **ret_val);
```

- Wait for the termination of a specific thread. Return immediately if thread has already ended.

```
@param [in]  thread      Thread ID to wait for terminate
@param [out] ret_val     Return value of terminated thread
@return                  0 if thread join succeeds
```

Pthreads API – pthread_attr_detachstate

```
int pthread_attr_setdetachstate(
      pthread_attr_t *attr, int detachstate);
int pthread_attr_getdetachstate(
      pthread_attr_t *attr, int* detachstate);
```

- set and get detach state of thread attributes object

```
@param[in]       attr          Thread attributes object
@param[in, out]  state         PTHREAD_CREATE_JOINABLE / PTHREAD_CREATE_DETACHED
@return                        0 on success; on error, they return a nonzero error number
```

Pthreads API – pthread_exit

```
void pthread_exit(void *ret_val);
```

- Exit the calling thread. *return* of the thread function is the implicit call of pthread_exit.

```
@param [in]  ret_val      Return value of the thread.
                          Parent thread can collect this with pthread_join.
```

Pthreads API – pthread_self

```
pthread_t pthread_self(void);
```

- Return the thread ID of calling thread.

```
@return                 Thread ID of the calling thread
```

Example1

< prac_basic.cpp >

```cpp
#include  <stdio.h>
#include <pthread.h>

#define NUM_THREADS 5

void* thread_func(void* arg) {
  pthread_t ret_value;

  ret_value = pthread_self();

  pthread_exit((void*)ret_value);
}


int main() {
  pthread_t threads[NUM_THREADS];

  for (int i = 0; i < NUM_THREADS; ++i) {
    if (pthread_create(&threads[i], NULL, thread_func, NULL) < 0) {
      printf("error: pthread_create failed!\n");
      return 0;
    }
  }

  pthread_t ret;
  for (int i = 0; i < NUM_THREADS; ++i) {
    pthread_join(threads[i], (void**)&ret);
    printf("thread %ld: returned thread id -> %ld\n", threads[i], ret);
  }
  return 0;
}
```

< Result >

```
~/TA/MC2021 g++ prac_basic.cpp -o prac_basic -lpthread

~/TA/MC2021 ./prac_basic
thread 139797466670848: returned thread id -> 139797466670848
thread 139797458278144: returned thread id -> 139797458278144
thread 139797449885440: returned thread id -> 139797449885440
thread 139797441492736: returned thread id -> 139797441492736
thread 139797360277248: returned thread id -> 139797360277248
```

Example2

< prac_attr.cpp >

```cpp
#include  <stdio.h>
#include <pthread.h>

#define NUM_THREADS 5

void* thread_func(void* arg) {
  long dump[1024 * 1024] = {0,};
  pthread_t ret_value;

  ret_value = pthread_self();

  pthread_exit((void*)ret_value);
}


int main() {
  pthread_t threads[NUM_THREADS];

  for (int i = 0; i < NUM_THREADS; ++i) {
    if (pthread_create(&threads[i], &attr, thread_func, NULL) < 0) {
      printf("error: pthread_create failed!\n");
      return 0;
    }
  }

  pthread_t ret;
  for (int i = 0; i < NUM_THREADS; ++i) {
    pthread_join(threads[i], (void**)&ret);
    printf("thread %ld: returned thread id -> %ld\n", threads[i], ret);
  }
  return 0;
}
```

< Result >

```
~/TA/MC2021  g++ prac_attr.cpp -o prac_attr -lpthread

~/TA/MC2021  ./prac_attr
[1]    16103 segmentation fault (core dumped)  ./prac_attr
```

Example2 (Cont.)

< prac_attr_mod.cpp >

```
int main() {
  pthread_t threads[NUM_THREADS];
  pthread_attr_t attr;
  size_t stacksize;

  pthread_attr_init(&attr);
  pthread_attr_setstacksize(&attr, 16 * 1024 * 1024);

  for (int i = 0; i < NUM_THREADS; ++i) {
    if (pthread_create(&threads[i], &attr, thread_func, NULL) < 0) {
      printf("error: pthread_create failed!\n");
      return 0;
    }
  }

  pthread_attr_destroy(&attr);

  pthread_t ret;
  for (int i = 0; i < NUM_THREADS; ++i) {
    pthread_join(threads[i], (void**)&ret);
    printf("thread %ld: returned thread id -> %ld\n", threads[i], ret);
  }
  return 0;
}
```

< Result >

```
~/TA/MC2021 ./prac_attr
thread 139927260370688: returned thread id -> 139927260370688
thread 139927243589376: returned thread id -> 139927243589376
thread 139927226808064: returned thread id -> 139927226808064
thread 139927210026752: returned thread id -> 139927210026752
thread 139927193245440: returned thread id -> 139927193245440
```

Example3

< prac_pthread.cpp >

```
 1 #include <stdio.h>
 2 #include <pthread.h>
 3
 4 #define NUM_THREADS     10
 5 #define NUM_INCREMENT   1000000
 6
 7 long cnt_global = 0;
 8
 9 void* thread_func(void* arg) {
10     long cnt_local = 0;
11
12     for (int i = 0; i < NUM_INCREMENT; i++) {
13         cnt_global++;   // increase global value
14         cnt_local++;    // increase local value
15     }
16
17     return (void*)cnt_local;
18 }
```

Example3 (continue..)

```
20 int main(void) {
21     pthread_t threads[NUM_THREADS];
22
23     // create threads
24     for (int i = 0; i < NUM_THREADS; i++) {
25         if (pthread_create(&threads[i], 0, thread_func, NULL) < 0) {
26             printf("error: pthread_create failed!\n");
27             return 0;
28         }
29     }
30
31     // wait the threads end
32     long ret;
33     for (int i = 0; i < NUM_THREADS; i++) {
34         pthread_join(threads[i], (void**)&ret);
35         printf("thread %ld: local count -> %ld\n", threads[i], ret);
36     }
37     printf("global count -> %ld\n", cnt_global);
38
39     return 0;
40 }
```

Example3 (continue..)

< Result >

```
jongbin@multicore-96:~/TA/Multicore$ g++ prac_pthread.cpp -o prac_pthread -lpthread
jongbin@multicore-96:~/TA/Multicore$ ./prac_pthread
thread 140003858446080: local count -> 1000000
thread 140003841668864: local count -> 1000000
thread 140003833276160: local count -> 1000000
thread 140003824883456: local count -> 1000000
thread 140003816490752: local count -> 1000000
thread 140003808098048: local count -> 1000000
thread 140003799705344: local count -> 1000000
thread 140003791312640: local count -> 1000000
thread 140003782919936: local count -> 1000000
thread 140003774527232: local count -> 1000000
global count -> 1401495
```

Example3 (continue..)

```
 9 void* thread_func(void* arg) {
10     long cnt_local = 0;
11
12     for (int i = 0; i < NUM_INCREMENT; i++) {
13         cnt_global++;    // increase global value
14         cnt_local++;     // increase local value
15     }
16
17     return (void*)cnt_local;
18 }
```

Example3 (continue..)

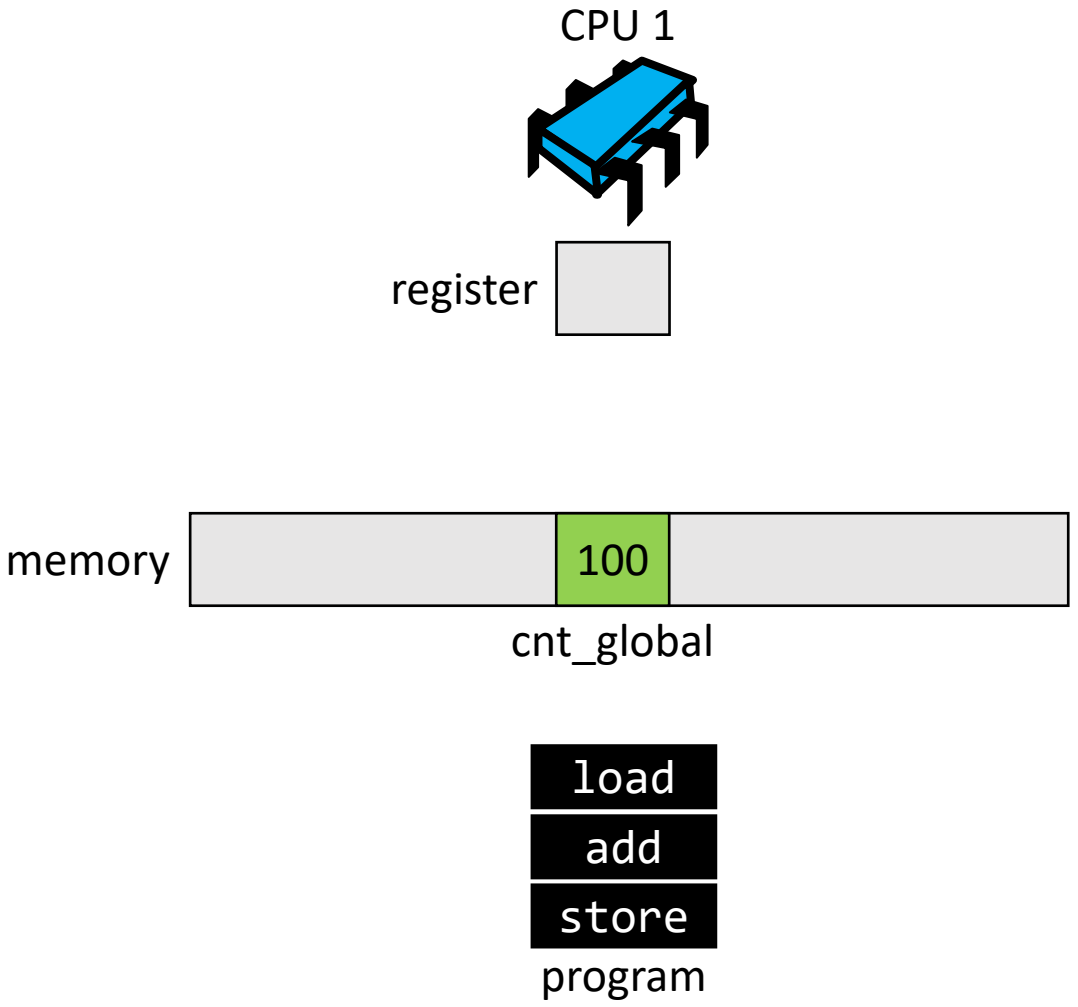
< assembly instructions for *cnt_global++* in C code >

```
$ g++ -S prac_pthread.cpp
```

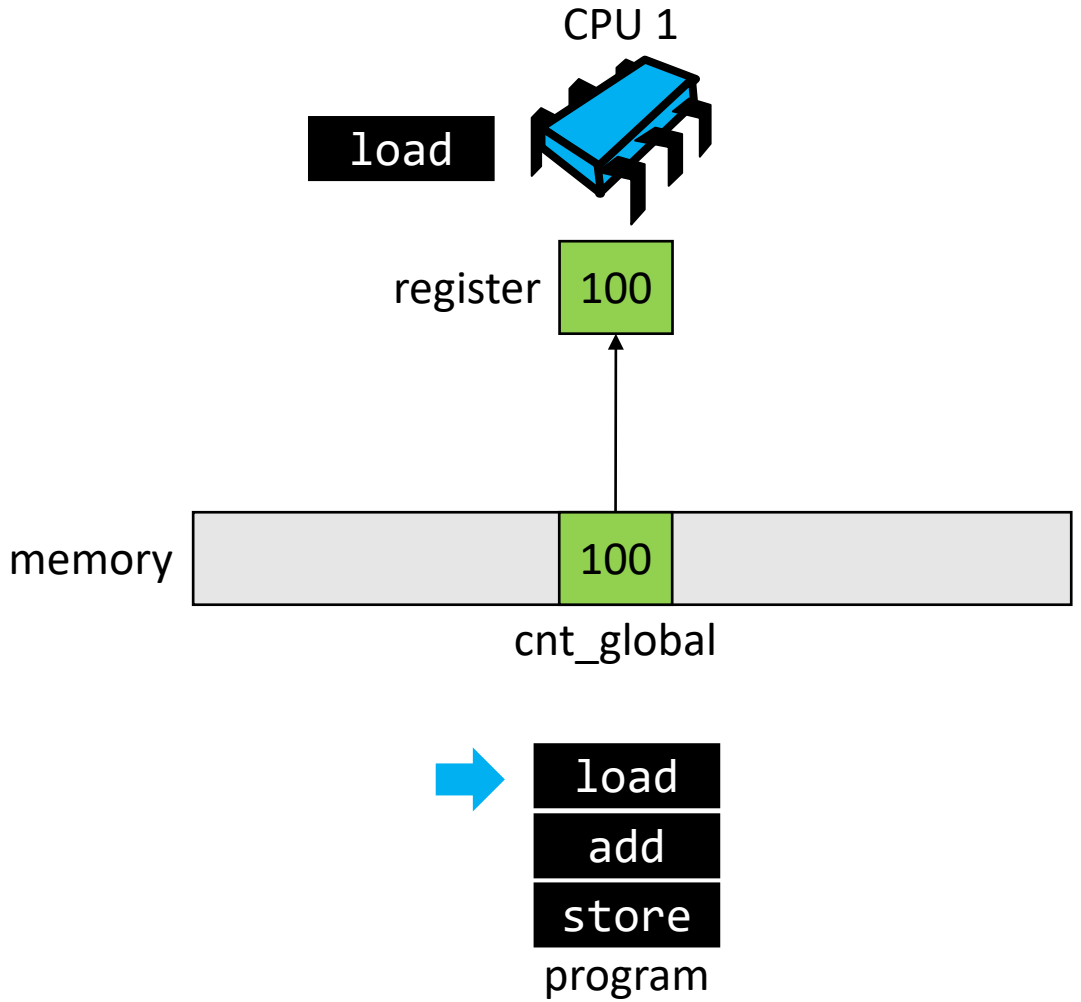
```
23 .L3:
24     cmpl    $999999, -12(%rbp)
25     jg  .L2
26     movq    cnt_global(%rip), %rax
27     addq    $1, %rax
28     movq    %rax, cnt_global(%rip)
29     addq    $1, -8(%rbp)
30     addl    $1, -12(%rbp)
31     jmp .L3
32 .L2:
33     movq    -8(%rbp), %rax
```

load
add
store

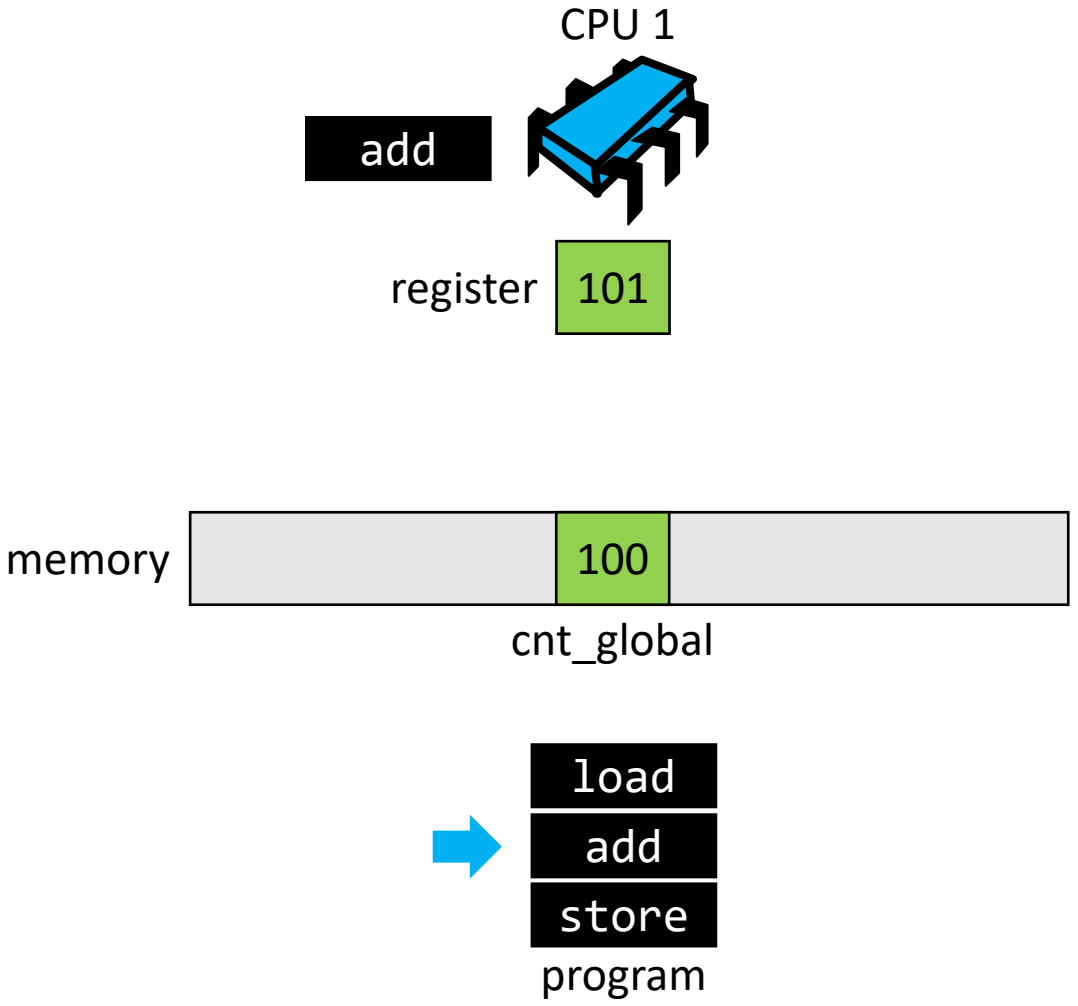
Example: Single thread



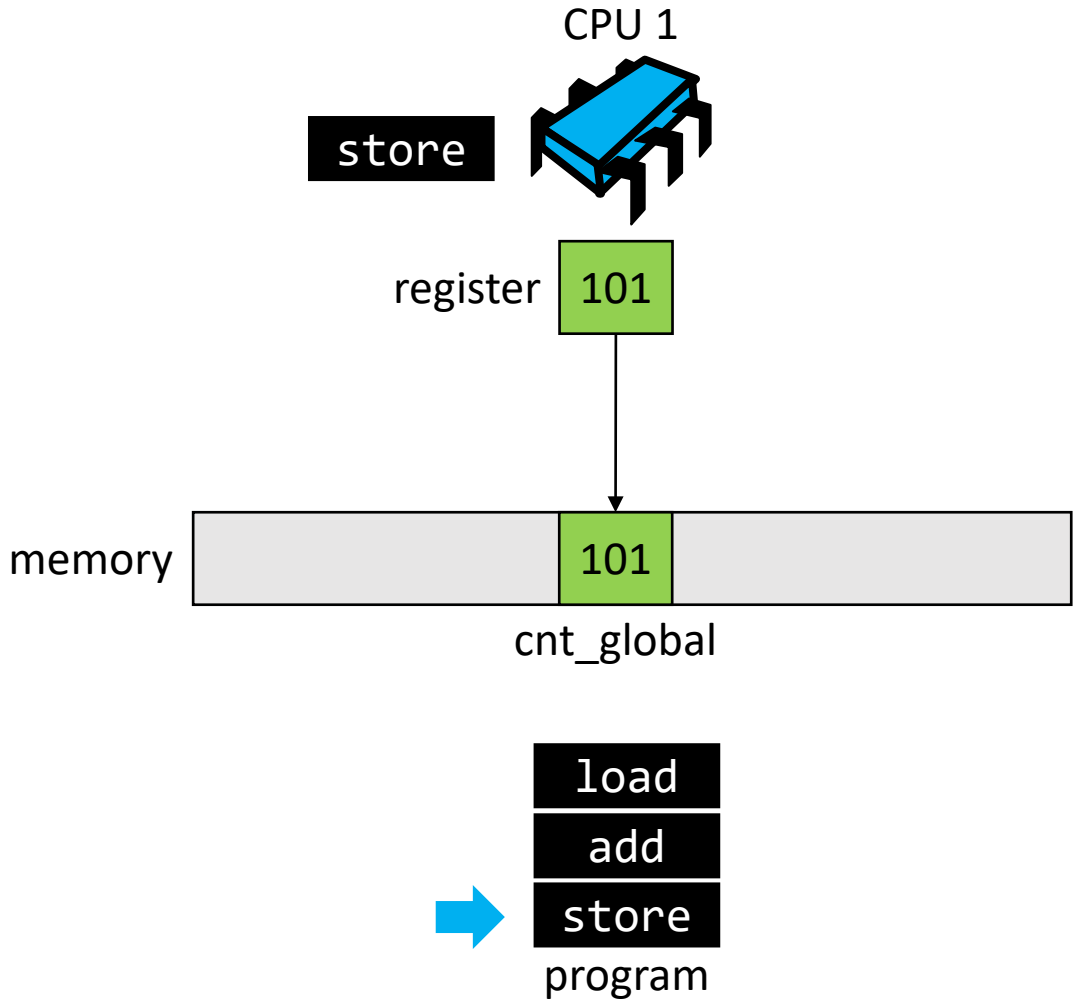
Example: Single thread



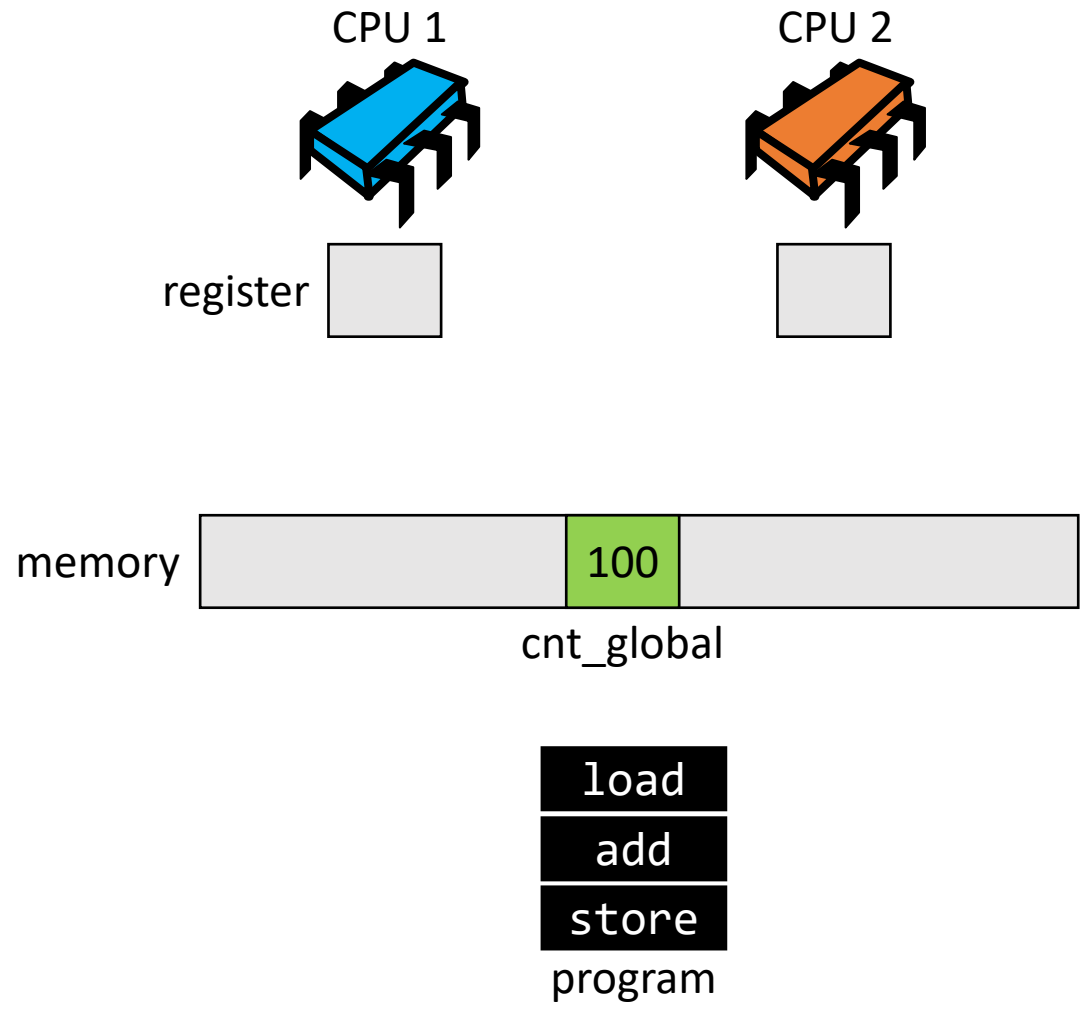
Example: Single thread



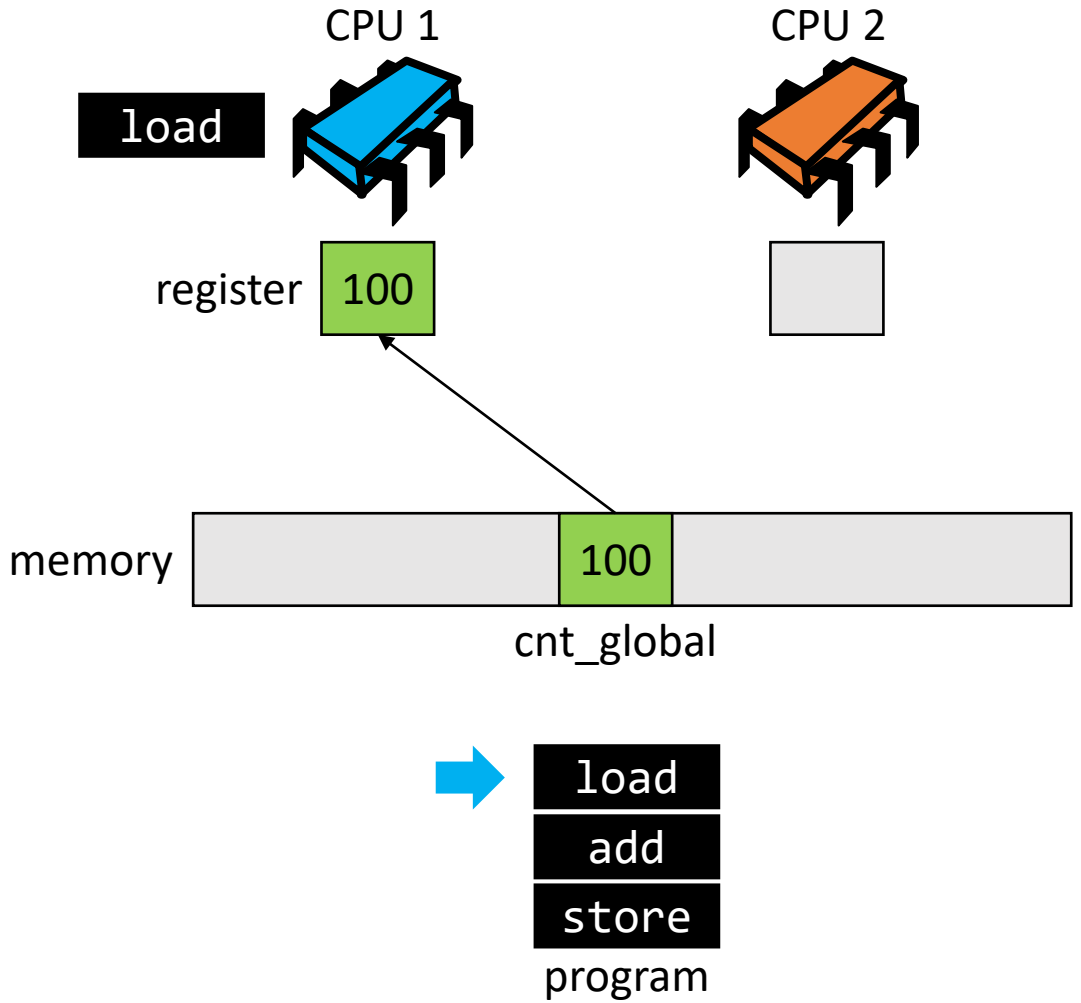
Example: Single thread



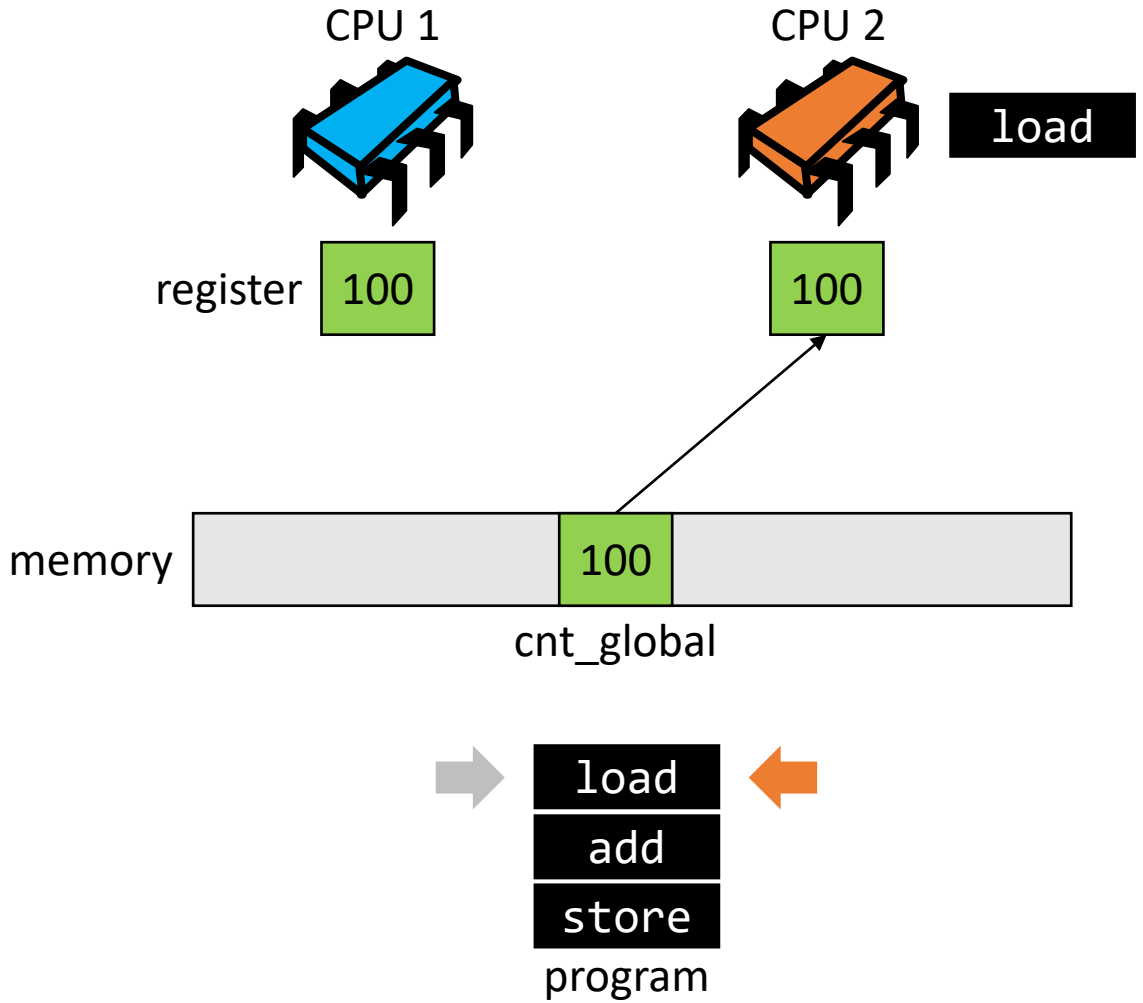
Example: Two threads



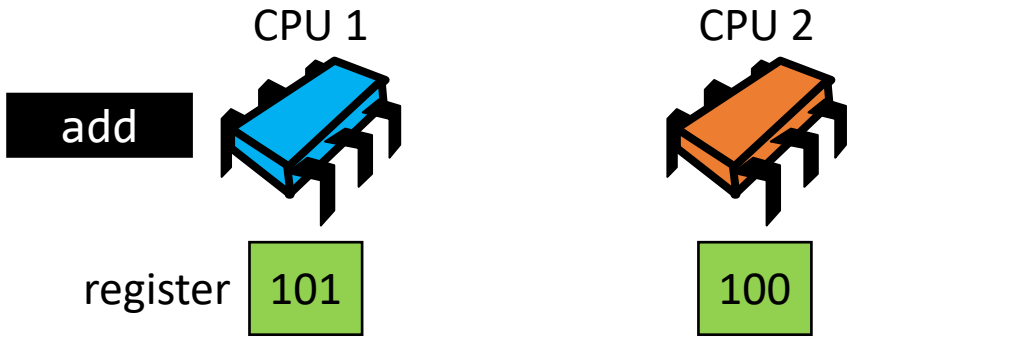
Example: Two threads

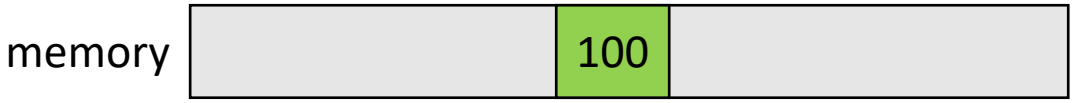


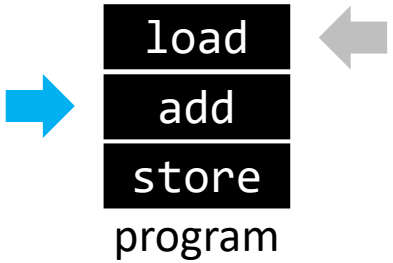
Example: Two threads



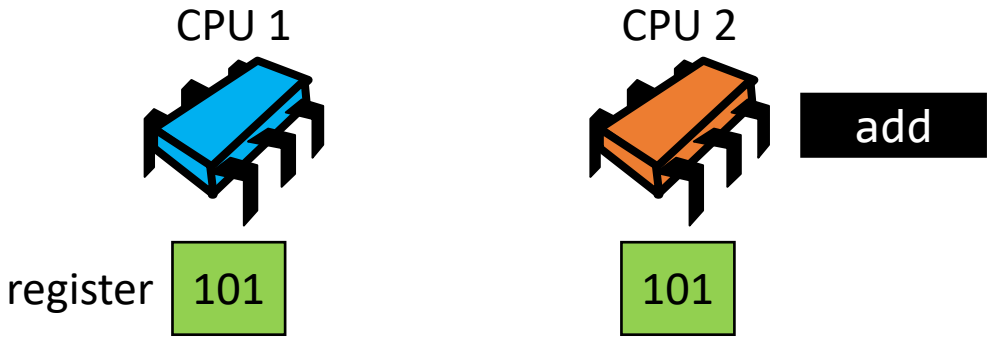
Example: Two threads

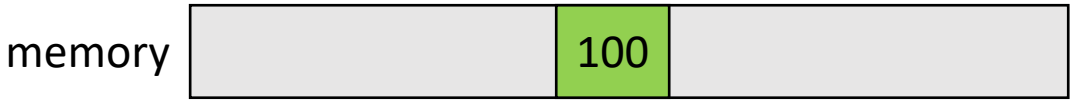




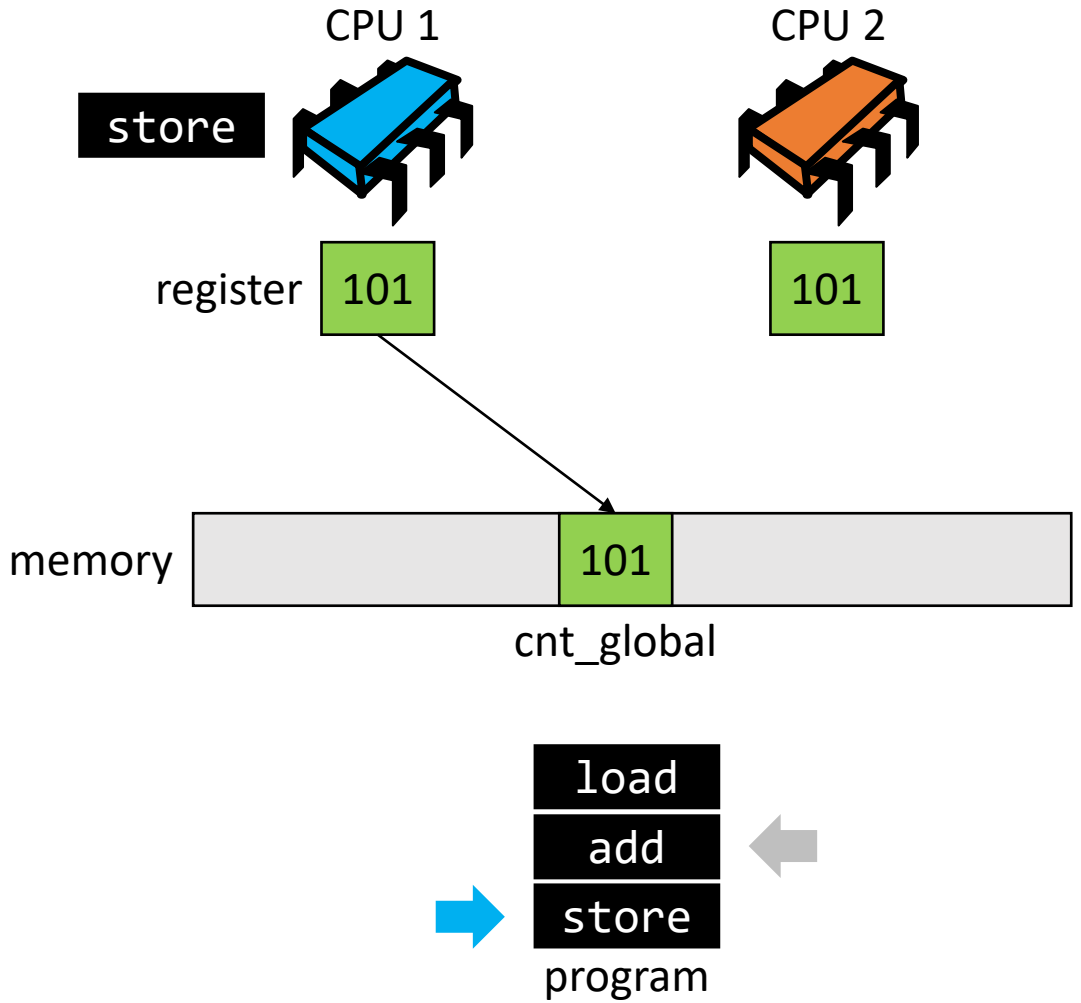


Example: Two threads

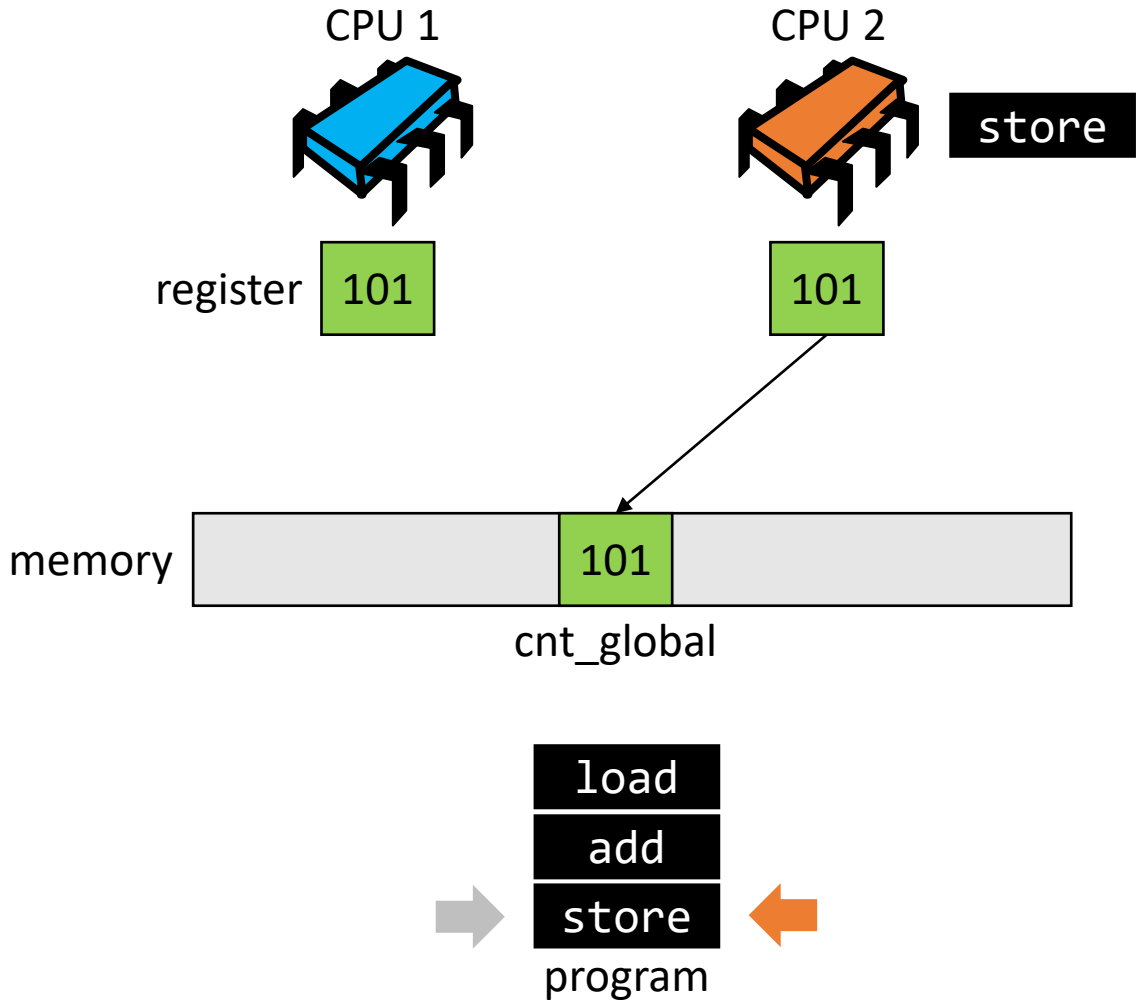




Example: Two threads



Example: Two threads



Thank You