Атомарные операции в ARMv6 и ARMv7 на примере: LDREX/STREX. Реализация в стандарте C11

Содержание:

- Проблематика (зачем?)
- Как работают LDREX/STREX
- Отличие от SWP(SWAP)
- Пример атомарной операции
- Сравнение с Mutex, Spin_lock
- Реализация в стандарте C11

Проблематика (зачем?):

```
void* thread1(void *arg){
    while(1){
        If(...)
            Errors |= Error_Flag1;

        //some code
    }
    return NULL;
}
```

```
void* thread2(void *arg){
    while(1){
        If(...)
            Errors |= Error_Flag2;

        //some code
    }
    return NULL;
}
```

Проблематика (решение 1):

```
void* thread1(void *arg){
    while(1){
        If(...){
            mutex_lock();
            Errors |= Error_Flag1;
            mutex_unlock();
        }
        //some code
    }
    return NULL;
}
```

```
void* thread2(void *arg){
    while(1){
        If(...){
            mutex_lock();
            Errors |= Error_Flag2;
            mutex_unlock();
        }
        //some code
    }
    return NULL;
}
```

Проблематика (решение 2):

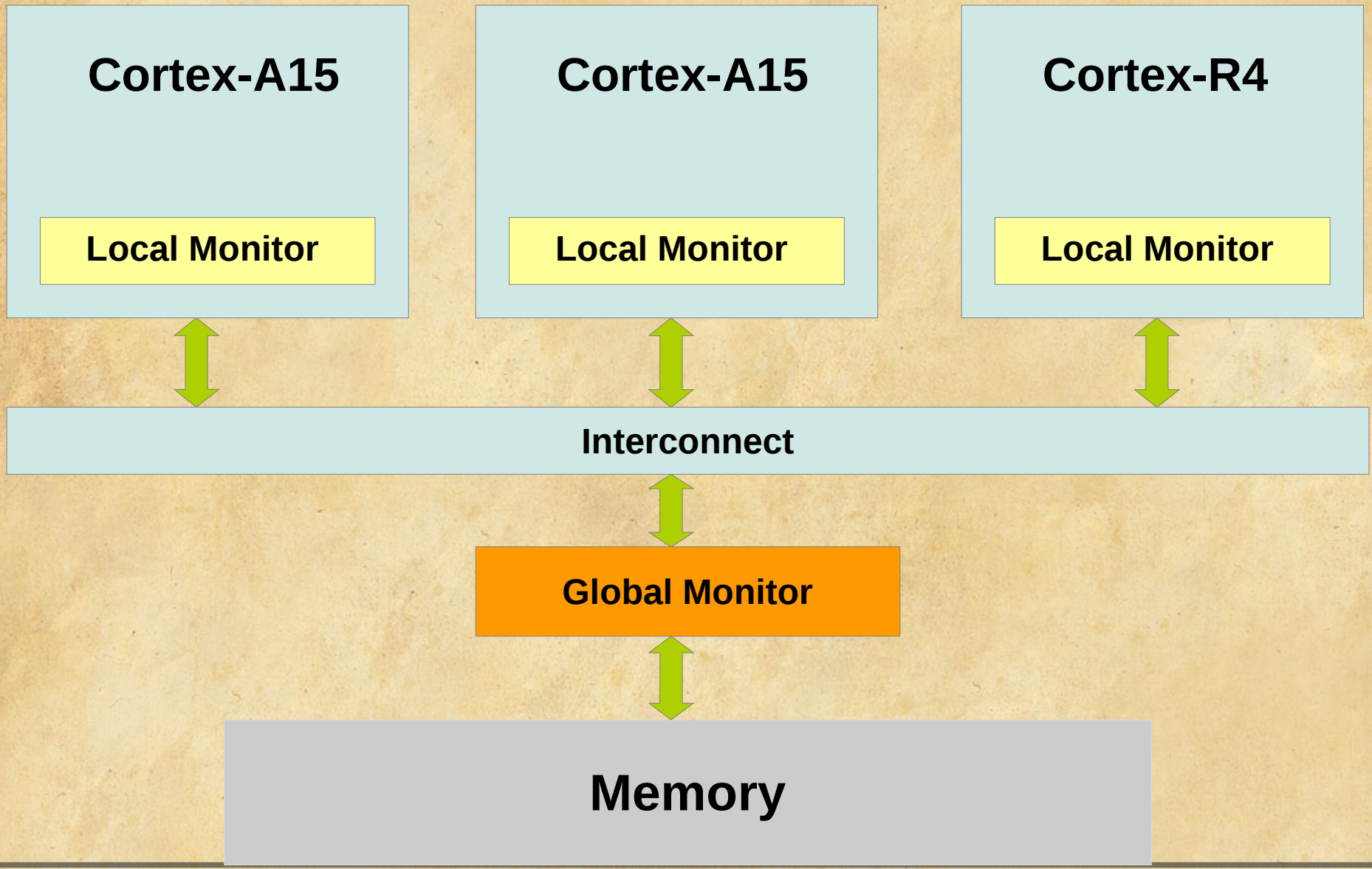
```
void* thread1(void *arg){
    while(1){
        If(...){
        atomic_or(Errors, Error_Flag1);
        }
        //some code
    }
    return NULL;
}
```

```
void* thread2(void *arg){
    while(1){
        If(...){
        atomic_or(Errors, Error_Flag2);
        }
        //some code
    }
    return NULL;
}
```

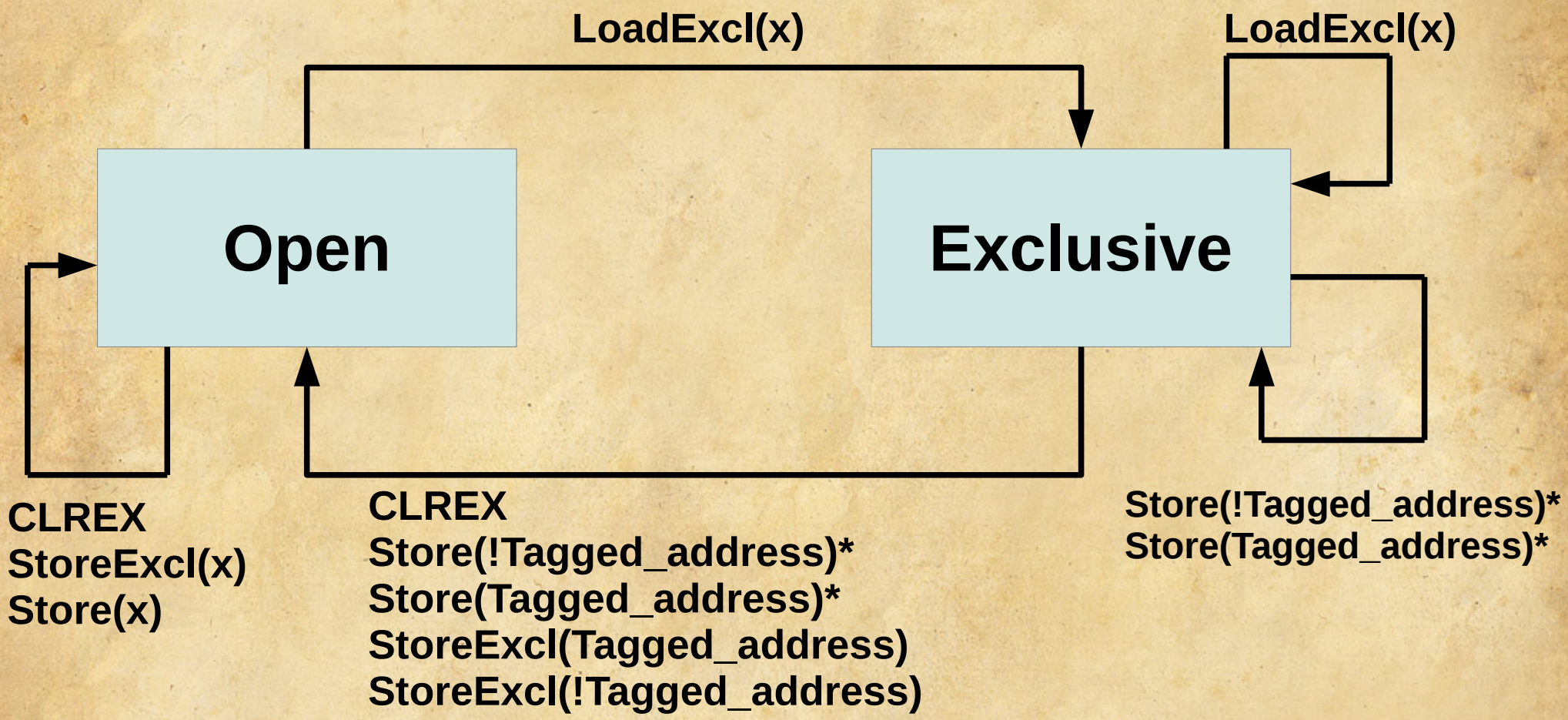
Как работают LDREX/STREX:

Core1	Core2	SUCCESS
LDREX(x)		
STREX(x)		**Yes**
	LDREX(x)	
LDREX(x)		
	STREX(x)	**Yes**
STREX(x)		**No**
LDREX(x)		
STR(x)		
STREX(x)		**Yes**

Local and Global Exclusive Monitor:



Local monitor state machine diagram:



Load and Store Register Exclusive:

Syntax:

LDREX{**cond**} **Rt**, [**Rn** {, #offset}]
STREX{**cond**} **Rd**, **Rt**, [**Rn** {, #offset}]
LDREXB{**cond**} **Rt**, [**Rn**]
STREXB{**cond**} **Rd**, **Rt**, [**Rn**]
LDREXH{**cond**} **Rt**, [**Rn**]
STREXH{**cond**} **Rd**, **Rt**, [**Rn**]
LDREXD{**cond**} **Rt**, **Rt2**, [**Rn**]
STREXD{**cond**} **Rd**, **Rt**, **Rt2**, [**Rn**]
CLREX

Cond is an optional condition code.
Rd is the destination register for the returned status.
Rt is the register to load or store.
Rt2 is the second register for doubleword loads or stores.
Rn is the register on which the memory address is based.

Load and Store Register Exclusive:

	ARMv6	ARMv6K	ARMv7	ARMv7-M
LDREX	ок	ок	ок	ок
STREX	ок	ок	ок	ок
LDREXB	---	ок	ок	ок
STREXB	---	ок	ок	ок
LDREXH	---	ок	ок	ок
STREXH	---	ок	ок	ок
LDREXD	---	ок	ок	---
STREXD	---	ок	ок	---
CLREX	---	ок	ок	ок

Инструкция SWP:

Syntax

SWP{**B**}{**cond**} **Rt**, **Rt2**, [**Rn**]

Cond is an optional condition code.

B If B is present, a byte is swapped. Otherwise, a 32-bit word is swapped.

Rt is the destination register. Rt must not be PC.

Rt2 is the source register. Rt2 can be the same register as Rt. Rt2 must not be PC.

Rn contains the address in memory. Rn must be a different register from both Rt and Rt2. Rn must not be PC.

Инструкция SWP:

```
LOCKED EQU 1          ; define value indicating

 LDR   r1, <addr>     ; load spin address
 LDR   r0, =LOCKED    ; preload "locked" value

spin_lock
SWP    r0, r0, [r1]   ; swap register value with spin
CMP    r0, #LOCKED    ; if spin was locked already
BEQ    spin_lock      ; retry
DMB    SY             ; if system is SMP
```

Реализация Spin lock:

spin_lock

```
MOV       R1,  #1
loop:
LDREX     R2,  [R0]
CMP       R2,  #0
STREXeq   R2,  R1,  [R0]
CMPeq     R2,  #0
Bne       loop
DMB       SY
```

spin_unlock

```
MOV       R1,  #0
DMB       SY
STR       R1, [R0]
```

atomic_or:

```
void  atomic_or(int *obj; int val);

atomic_or:

.try_or:
    LDREX   R2,  [R0]
    ORR     R12, R2,  R1
    STREX   R3,  R12, [R0]
    CMP     R3,  #0
    BNE     .try_or

    BX      LR
```

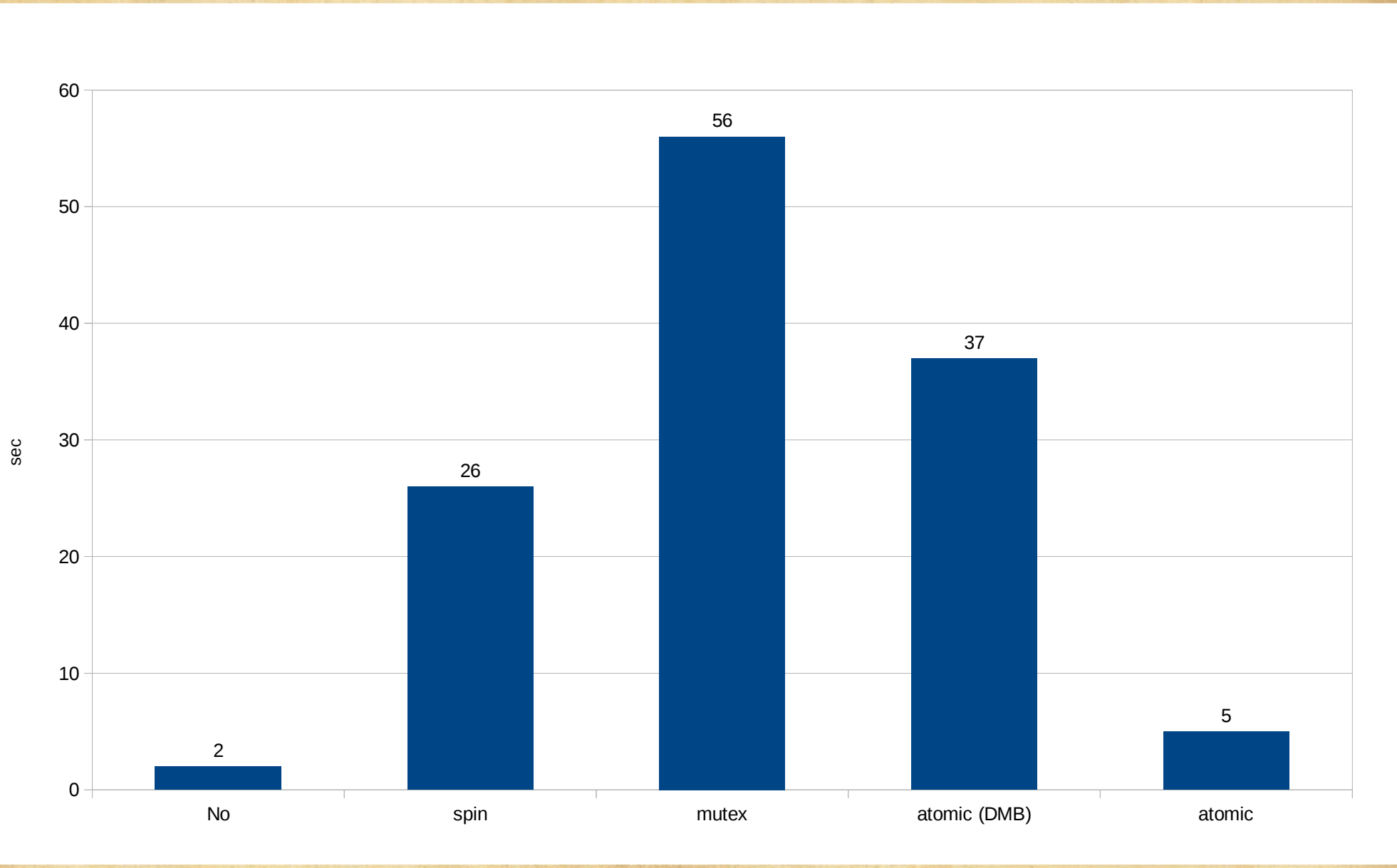
atomic_add (return):

```
int  atomic_add(int *obj; int val);

atomic_add:

.try_add:
    LDREX   R2,  [R0]
    ADD     R12, R2,  R1
    STREX   R3,  R12, [R0]
    CMP     R3,  #0
    BNE     .try_add

    MOV     R0,  R2
    BX      LR
```

atomic_add (return):

```
int  atomic_add(int *obj; int val);

atomic_add:
    DMB         SY                      // for unlock
.try_add:
    LDREX   R2,  [R0]
    ADD       R12, R2,  R1
    STREX   R3,  R12, [R0]
    CMP       R3,  #0
    BNE       .try_add
    DMB         SY                      // for lock
    MOV       R0,  R2
    BX          LR
```

Mutex vs atomic



Atomic in C11

```
#include <stdatomic.h>   // начиная  с  gcc 4.9.0
```

atomic_exchange(volatile A *object, C desired);

atomic_exchange_explicit(volatile A *object, C desired, memory_order order);

C **atomic**_fetch_**key**(volatile A *object, M operand);

C **atomic**_fetch_**key**_explicit(volatile A *object, M operand, memory_order order);

key	op	computation
add	+	addition
sub	-	subtraction
or	|	bitwise inclusive or
xor	/\	bitwise exclusive or
and	&	bitwise and

memory_order_relaxed	No
memory_order_consume	See Doc
memory_order_acquire	Lock
memory_order_release	Unlock
memory_order_acq_rel	See Doc
memory_order_seq_cst	See Doc

Вопросы...