1. **Чем отличается микроконтроллер от микропроцессора?**

Микроконтро́ллер — [микросхема](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D1%81%D1%85%D0%B5%D0%BC%D0%B0), предназначенная для управления [электронными](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%B8%D0%BA%D0%B0) [устройствами](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE).

Типичный микроконтроллер сочетает на одном кристалле функции [процессора](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) и [периферийных устройств](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B8%D1%84%D0%B5%D1%80%D0%B8%D0%B9%D0%BD%D0%BE%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE), содержит [ОЗУ](https://ru.wikipedia.org/wiki/%D0%9E%D0%97%D0%A3) и (или) [ПЗУ](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D1%82%D0%BE%D1%8F%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B7%D0%B0%D0%BF%D0%BE%D0%BC%D0%B8%D0%BD%D0%B0%D1%8E%D1%89%D0%B5%D0%B5_%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE). По сути, это однокристальный [компьютер](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80), способный выполнять относительно простые задачи.

Отличается от [микропроцессора](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BA%D1%80%D0%BE%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80) интегрированными в микросхему устройствами ввода-вывода, таймерами и другими периферийными устройствами.

1. **Какие тактовые частоты могут быть у ATmega8535?**

от 0 до 16 MГц

1. **Какие таймеры есть у ATmega8535?**

- два 8-разрядных таймера 0 и 2

- один 16-разрядный таймер 1

- программируемый сторожевой таймер с внутренним генератором

1. **Сколько прерываний есть у ATmega8535?**

21 прерывание: 4 внешних — INT0, INT 1, INT2, RESET и 17 внутренних.

1. **Как организована система приоритетов при обработке прерываний?**

Контроллер прерываний NVIC - приоритетный. Если случилось так, что несколько прерываний встали на ожидание, но не смогли пойти на выполнение (т.к., например, был глобальный запрет, либо выполнялось прерывание), то как только появится возможность первым пойдет то прерывание, чей номер в таблице векторов меньше.

За приоритет отвечает группа регистров IPRx всего их там двадцать (но вообще может быть до 60). В каждом регистре по четыре 8 битных поля, каждое поле отвечает за один вектор. От 0 (помним, что номера считаются от External Interrupt, исключая FIQ группу) до 80. В IRP0 лежат приоритеты с 0 по 3 вектор, в IRP1 с 4 по 7 и так далее до 80го.

Теоретически приоритетов может быть 256 штук. Но это только в теории, это возможности ядра в принципе. А конкретно в STM32F103 реально рулят только старшие 4ре бита. Образуя таким образом 16 уровней приоритетов. Чем меньше номер уровня, тем он главней. После RESET у всех 0000

6) **Как можно запретить (разрешить) сразу все прерывания?**

*Но для того, чтобы программа перешла в обработчик прерывания этого недостаточно, у STM32 надо разрешить его в 3-х местах:*

разрешить прерывания глобально

* разрешить прерывание в NVIC
* разрешить прерывание непосредственно в самой периферии

Запретить (разрешить) сразу все прерывания можно, выставляя соответствующие биты, создавая группы приоритетов. Для этого пользуются функциями CMSIS:

void NVIC\_SetPriorityGrouping(uint32\_t priority\_grouping) *// Задать группы/подгруппы приоритетов*

void NVIC\_EnableIRQ(IRQn\_t IRQn) *// Включить IRQn*

void NVIC\_DisableIRQ(IRQn\_t IRQn) *// Выключить IRQn*

uint32\_t NVIC\_GetPendingIRQ (IRQn\_t IRQn) *// Вернуть номер IRQn если оно в ожидании*

void NVIC\_SetPendingIRQ (IRQn\_t IRQn) *// Поставить IRQn в ожидание*

void NVIC\_ClearPendingIRQ (IRQn\_tIRQn) *// Убрать IRQn из очереди ожидания*

uint32\_t NVIC\_GetActive (IRQn\_t IRQn) *// Возвращает номер текущего активного прерывания*

void NVIC\_SetPriority (IRQn\_t IRQn, uint32\_t priority) *// Задать приоритет IRQn*

uint32\_t NVIC\_GetPriority (IRQn\_t IRQn) *// Считать приоритет IRQn*

*7)* **Как можно разрешить(запретить) прерывания по переполнению таймера 0?**

Таймеры 16 битные (то есть могут считать до 65535), умеют работать с инкрементальными энкодерами и датчиками Холла, несколько таймеров можно синхронизировать между собой. Есть прерывания на разные события.

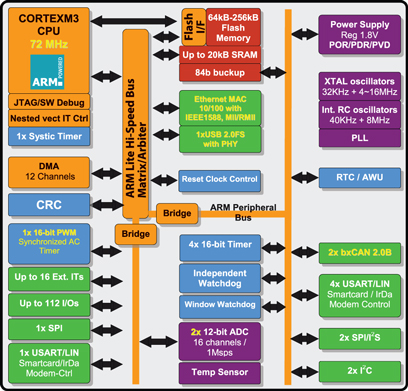
Разрешить прерывание по переполнению:

TIM\_ITConfig(TIM0, TIM\_IT\_Update, ENABLE);

Разрешить прерывание по переполнению:

TIM\_ITConfig(TIM0, TIM\_IT\_Update, DISABLE);

8) **Внутренняя структура МК?**



9) **Таймер 1. Режимы работы, количество прерываний, регистры?**

Таймеры с расширенным управлением занимают особое место среди прочих таймеров в микроконтроллерах STM. Эти таймеры имеют наиболее сложное устройство, в них реализовано наибольшее количество функций, что обеспечивает максимум возможностей при их использовании.

Перезагрузка счетчика формирует событие (прерывание). Частота его появления также может быть уменьшена счетчиком повторов (8 разрядов). Коэффициент деления задается в регистре повторов.

TIM1\_CCER - регистр разрешения захвата/сравнения

TIM1\_BDTR - break and dead-time register

TIM1\_CCR1 - регистр 1 захвата/сравнения

TIM1\_CCR2 - регистр 2 захвата/сравнения

TIM1\_CCR3 - регистр 3 захвата/сравнения

TIM1\_CCR4 - регистр 4 захвата/сравнения

TIM1\_CR1 - регистр управления 1

10) **Написать программу с использованием таймера 10, вырабатывающую на PORTA0 симметричное прямоугольное колебание с периодом 2мс.**

### Генерация сигнала из обработчика прерываний

*/\* USER CODE BEGIN 0 \*/*

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

static int val = 0;

if (htim->Instance==TIM1) *//check if the interrupt comes from TIM1*

{

HAL\_DAC\_SetValue(&hdac, DAC\_CHANNEL\_1, DAC\_ALIGN\_12B\_R, val);

val = val? 0: 4095;

}

}

*/\* USER CODE END 0 \*/*

...

*/\* USER CODE BEGIN 2 \*/*

HAL\_TIM\_Base\_Start\_IT(&htim1);

\_\_HAL\_DAC\_ENABLE(&hdac, DAC\_CHANNEL\_1);

*/\* USER CODE END 2 \*/*

void TIM1\_UP\_TIM10\_IRQHandler(void)

{

*/\* USER CODE BEGIN TIM1\_UP\_TIM10\_IRQn 0 \*/*

static int val = 0;

*/\* TIM Update event \*/*

\_\_HAL\_TIM\_CLEAR\_IT(&htim1, TIM\_IT\_UPDATE);

if (htim1.Instance==TIM1) {

\*(uint32\_t\*)(DAC\_BASE + 0x11BA8EFU) = val;

val = val? 0: 4095;

}

return;

*/\* USER CODE END TIM1\_UP\_TIM10\_IRQn 0 \*/*

HAL\_TIM\_IRQHandler(&htim1);

*/\* USER CODE BEGIN TIM1\_UP\_TIM10\_IRQn 1 \*/*

*/\* USER CODE END TIM1\_UP\_TIM10\_IRQn 1 \*/*

}

11) **Какой режим таймера 0 позволяет вырабатывать треугольные колебания, используя дополнительную интегрирующую цепочку?**

Режим1 - ШИМ-режим - до достижения максимального значения ($FF) вверх (суммирование), потом вниз (вычитание).

12) **Какие значения записаны в TCCR0 после сигнала RESET?**

Ноль.

13) **Что такое SPI и зачем он нужен?**

SPI ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Serial Peripheral Interface, SPI bus* — последовательный периферийный интерфейс, шина SPI) — последовательный синхронный стандарт передачи данных в режиме полного [дуплекса](https://ru.wikipedia.org/wiki/%D0%94%D1%83%D0%BF%D0%BB%D0%B5%D0%BA%D1%81_(%D1%82%D0%B5%D0%BB%D0%B5%D0%BA%D0%BE%D0%BC%D0%BC%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D0%B8)), предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии. SPI также иногда называют четырёхпроводным ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *four-wire*) интерфейсом.

В отличие от стандартного последовательного порта ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *standard serial port*), SPI является синхронным интерфейсом, в котором любая передача синхронизирована с общим тактовым сигналом, генерируемым ведущим устройством (процессором). Принимающая (ведомая) периферия синхронизирует получение битовой последовательности с тактовым сигналом. К одному последовательному периферийному интерфейсу ведущего устройства-микросхемы может присоединяться несколько микросхем. Ведущее устройство выбирает ведомое для передачи, активируя сигнал «выбор кристалла» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *chip select*) на ведомой микросхеме. Периферия, не выбранная процессором, не принимает участия в передаче по SPI.

14) **Как инициировать передачу байта в SPI?**

Ведущее устройство инициирует цикл связи установкой низкого уровня на выводе выбора подчиненного устройства (SS) того устройства, с которым необходимо установить соединение. При низком уровне сигнала SS:

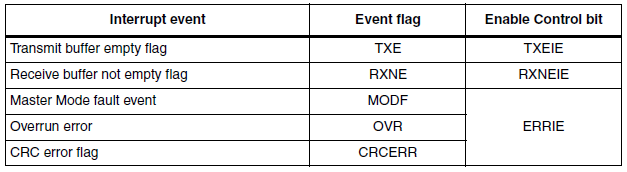
* схемотехника ведомого устройства находится в активном состоянии;
* вывод MISO переводится в режим «выход»;
* тактовый сигнал SCLK от ведущего устройства воспринимается ведомым и вызывает считывание на входе MOSI значений передаваемых от ведущего битов и сдвиг регистра ведомого устройства.

Подлежащие передаче данные ведущее и ведомое устройства помещают в сдвиговые регистры. После этого ведущее устройство начинает генерировать импульсы синхронизации на линии SCLK, что приводит к взаимному обмену данными. Передача данных осуществляется бит за битом от ведущего по линии MOSI и от ведомого по линии MISO. Передача осуществляется, как правило, начиная со старших битов, но некоторые производители допускают изменение порядка передачи битов программными методами. После передачи каждого пакета данных ведущее устройство, в целях синхронизации ведомого устройства, может перевести линию SS в высокое состояние.

15) **Сколько прерываний и регистров ввода-вывода принадлежит SPI?**

Модуль SPI имеет 9 регистров.

1. SPI\_CR1 – первый управляющий регистр;
2. SPI\_CR2 – второй управляющий регистр;
3. SPI\_SR – регистр статуса, содержит флаги;
4. SPI\_DR – регистр данных;
5. SPI\_CRCPR – регистр,содержащий полином для вычисления CRC;
6. SPI\_RXCRCR – регистр,содержащий CRC принятых данных;
7. SPI\_TXCRCR – регистр,содержащий CRC передаваемых данных;



16) **Порт А. Сколько прерываний и регистров ввода-вывода принадлежит порту А? Назначение этих регистров?**

Конфигурация выводов задаётся в регистрах GPIOx\_CRL, GPIOx\_CRH, в этих регистрах для конфигурации каждого вывода отведено 4 бита

17) **Сторожевой таймер и особенность его работы?**

* Independent watchdog (IWDG) – независимый сторожевой таймер, программист устанавливает значение с которого таймер начинает считать вниз. Если счётчик доходит до нуля – он отправляет сигнал сброса. Для того что бы этого не происходило необходимо постоянно восстанавливать регистр со временем отсчёта. Таким образом, если наша программа зависла – устройство не превратиться в кирпич.
* Windowed Watchdog (WWDG) – оконный сторожевой таймер, работает аналогичным способом что и IWDG, однако кроме верхней границы, есть и нижняя. Если мы знаем минимальное время выполнения какой-то операции, то можем добавить условие – если счётчик восстановлен раньше срока, то вероятно что-то пошло не так и нужно послать сигнал сброса.

18) **Однопроводной интерфейс. Как выглядят физический 0 и 1?**

1-Wire – протокол передачи данных в обе стороны по одному проводу. МК — ведущее устройство на проводе. Обмен информацией ведётся по временным промежуткам — тайм-слотам(60мкс). Один тайм-слот — 1 бит. 5 команд — Запись 0. Запись 1, Чтение, Сброс, Присутствие. Запись 1 — НИЗКИЙ уровень в течение 1-15 мкс. В течении оставшегося свободного времени тайм-слота освобождает линию. Запись 0 — НИЗКИЙ уровень не меньше 60 мкс, но не больше 120 мкс. (НИЗКИЙ уровень меньше 0,8В)

19) **Какова максимальная скорость передачи?**

Скорость обмена по SPI определяет *«Baud rate generator»*, который задает частоту следования тактовых импульсов. Для этого предназначены разряды BR0,BR1 и BR2 регистра SPI\_CR1. Три разряда предполагают наличие восьми значений скорости. Можем оценить скорость обмена. Возьмем наихудший случай — STM32F100 с его «крейсерской» тактовой частотой 24MHz. Максимальная скорость модуля SPI при этом получится 24000000 / 2 = 12 000 000 бод(1 бод — это 1 бит в секунду). Один байт можно передать меньше чем за 0.7 микросекунды.

20) **Что такое серийный номер и какова его структура?**

Уникальный идентификатор (Unique ID) — уникальный номер устройства, позволяющий отличать его от других устройств.

Регистр «уникальный идентификатор устройства» (96 бит) обеспечивает ссылочный номер, который является уникальным для любого

устройства и в любом контексте. Эти биты не могут быть изменены пользователем.

Для использования Unique ID, необходимо в программу добавить следующий код:

volatile uint32\_t \*UniqueID = (uint32\_t \*)0x1FFFF7E8;

volatile uint32\_t \_\_UniqueID[3];

\_\_UniqueID[0] = UniqueID[0];

\_\_UniqueID[1] = UniqueID[1];

\_\_UniqueID[2] = UniqueID[2];

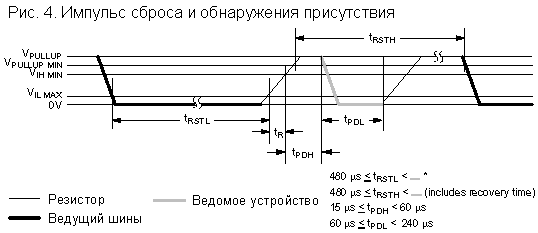
21) **Какая команда позволяет MASTER определить номера всех SLAVE в сети?**

Poll

Search ROM

Alarm Search

22) **Как выглядит сигнал сброса в сети MicroLAN?**



23) **Почему после сигнала RESET все остальные прерывания запрещены?**

После RESET в регистре флагов в седьмом разряде - 0.

24) **Приведите пример использования разряда T в регистре SREG?**

Т — для временного хранения бита Пример: перемещение второго бита регистра r1 в четвертый бит регистра r0: bst r1,2 - записать bld r0,4 - считать

25) **Как запрограммировать предварительный делитель таймера 0?**

**TIMx\_PSC** – входной предварительный делитель частоты, его значение можно рассчитать по формуле

**F = fCK\_PSC / (PSC[15:0] + 1)**

* **F** – частота с которой тактируется таймер
* **fCK\_PSC** — частота таймера до делителя
* **PSC[15:0]** — значение регистра PSC

26) **Чем сигнальный процессор отличается от микроконтроллера?**

Микроконтроллеры, обладающие больши́ми вычислительными возможностями -- цифровые сигнальные процессоры, применяющиеся для обработки большого потока данных в реальном времени (например, аудио-, видеопотоков).

27) **Как организовать вложенные прерывания?**

NVIC автоматически сохраняет данные при выполнении вложенных прерываний.

### Контроллер вложенных прерываний (NVIC)

Эта часть отвечает за генерацию прерываний на различные события: внешние — изменение логического уровня на входе ножки, пробуждение из режима сна; и внутренние — завершение приёма/отправки данных, переполнение счётчика таймера и т.п. Контроллер поддерживает до 240 прерываний и до 256 уровней приоритета, причём вход в обработчик прерывания занимает 12 тактов (сохранение стекового фрейма и регистров) и прерывания могут быть вложенными: если во время обработки прерывания возникает прерывание с меньшим приоритетом, то второе будет обработано через 6 тактов после обработки первого. Кроме того, существуют немаскируемые прерывания — NMI (**N**on-**M**asked **I**nterrupts), которые невозможно сбросить, не обработав, и которые прерывают выполнение программы независимо от каких-либо условий. Такие прерывания генерируются при сбое внешнего источника тактирования (кварца, керамического резонатора) и при обнаружении некорректной инструкции.

28) **Когда меняется порог в режиме 3 таймера 0?**

Особенность режима – двойная буферизация в регистр OCR0: записываемое число сохраняется в специальном буферном регистре. Изменение содержимого регистра порога меняется только при достижении счётчиком TNCT0 максимума в $FF. Благодаря этому исключается появление на выходе OC0 неправильных импульсов, глюков, которые были бы неизбежны при непосредственной записи в регистр порога без задержки.

29) **Зачем в программе нужно устанавливать начальное значение StackPointer и чему это значение должно быть равно?**

In this example, the stacks are located at stack\_base:

**Example 6. Initializing stack pointers**

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; This example does not apply to ARMv6-M and ARMv7-M profiles

; \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Len\_FIQ\_Stack EQU 256

Len\_IRQ\_Stack EQU 256

stack\_base DCD 0x18000

;

Reset\_Handler

; stack\_base could be defined above, or located in a scatter file

LDR R0, stack\_base ;

; Enter each mode in turn and set up the stack pointer

MSR CPSR\_c, #Mode\_FIQ:OR:I\_Bit:OR:F\_Bit ; Interrupts disabled

MOV sp, R0

SUB R0, R0, #Len\_FIQ\_Stack

MSR CPSR\_c, #Mode\_IRQ:OR:I\_Bit:OR:F\_Bit ; Interrupts disabled

MOV sp, R0

SUB R0, R0, #Len\_IRQ\_Stack

MSR CPSR\_c, #Mode\_SVC:OR:I\_Bit:OR:F\_Bit ; Interrupts disabled

MOV sp, R0

; Leave processor in SVC mode

The stack\_base symbol can be a hard-coded address, or it can be defined in a separate assembler source file and located by a scatter-loading description file.

30) **Режим 0 таймера 0**

Режим 0 – Normal. Счётчик TCNT0 - простой суммирующий счётчик. По каждому импульсу тактового сигнала, поступающего с выхода предварительного делителя, его содержимое увеличивается на 1. При переходе TCNT0 через максимум 0xFF возникает переполнение, счёт продолжается с 0x0, флаг переполнения TOV0 устанавливается в единицу. Как только TCNT0 = OCR0, то флаг прерывания OCF0 в регистре TIFR выставляется в 1 и (если разрешено) генерируется соответствующее прерывание.

31) **Режим 1 таймера 0**

Режим 1 – Phase Correct PWN – режим с точной фазой. В нём TCNT0 функционирует как реверсивный счётчик. На каждом сигнале с предварительного делителя значение TCNT0 изменяется на 0x1 в пределах от 0x0 до 0xFF, потом происходит смена направления, а потому на каждом сигнале происходит вычитание, пока значение не достигнет 0x0. Тогда происходит смена направления счёта и устанавливается флаг прерывания TOV0 регистра TIFR. Каждый раз при достижении значения OCR0 устанавливается флаг OCF0, генерируется соответствующее прерывание (если оно разрешено) и изменяется состояние выхода OC0.

32) **Режим 2 таймера 0**

Режим 2 – CTC – режим счёта по модулю OCR0. TCNT0 движется в пределах от 0x0 до OCR0. При достижении OCR0 TCNT0 сбрасывается в ноль, при этом выставляется флаг OCF0, генерируется соответствующее прерывание (если оно разрешено) и может измениться состояние вывода OC0.

33) **Режим 3 таймера 0**

Режим 3 - Fast PWM – быстродействующий ШИМ. Похож на режим 0, но выход OC0 (если он изменяется) изменяется и при достижении порога OCR0, и при достижении 0xFF. Это нужно для генерации ШИМ сигнала на выходе OC0.

34) **Можно ли писать в TCNT0 без остановки счёта?**

Разработчиками таймера приняты меры, чтобы запись и чтение можно было осуществлять без остановки счёта.

35) **Как можно остановить счёт в таймере 0?**

Записать в TCCR0 последние 3 нуля.

36) **Система прерываний микроконтроллера ATmega8535?**

21 прерывание. 4 внешних: INT0, INT1, INT2 и RESET. 17 внутренних, обслуживающих дополнительные блоки. Прерывания – некие внешние события, возникающие во время исполнения программы (нажатие кнопки или переполнение таймера). При обнаружении их МК вызывает обработчик прерывания соответствующего прерывания (программу обработки). Этот обработчик выполняет действие и возвращается в основную программу командой reti (Return from Interrupt). За обработку прерываний отвечает Interrupt Unit.