

Assembly VII

ECE 3710

I get no respect. The way
my luck is running, if I was a
politician I would be honest.

- Rodney Dangerfield

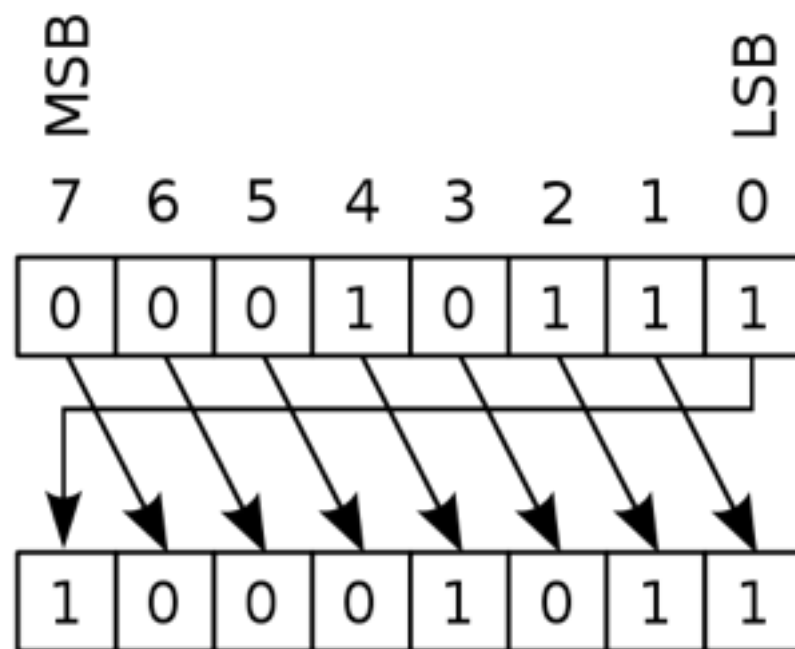
instructions:

1. shifts

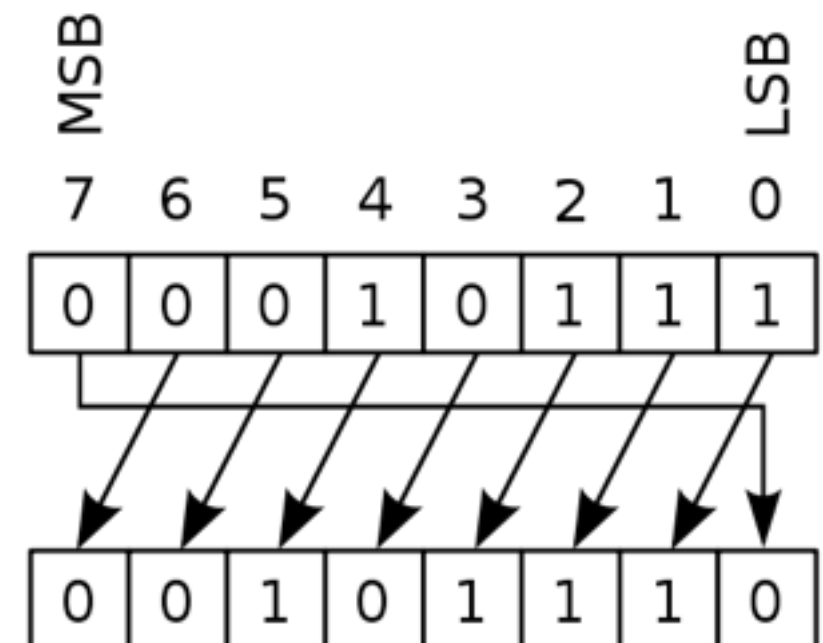
2. arithmetical

circular _(rotate) shift (not like C)

rotate right:

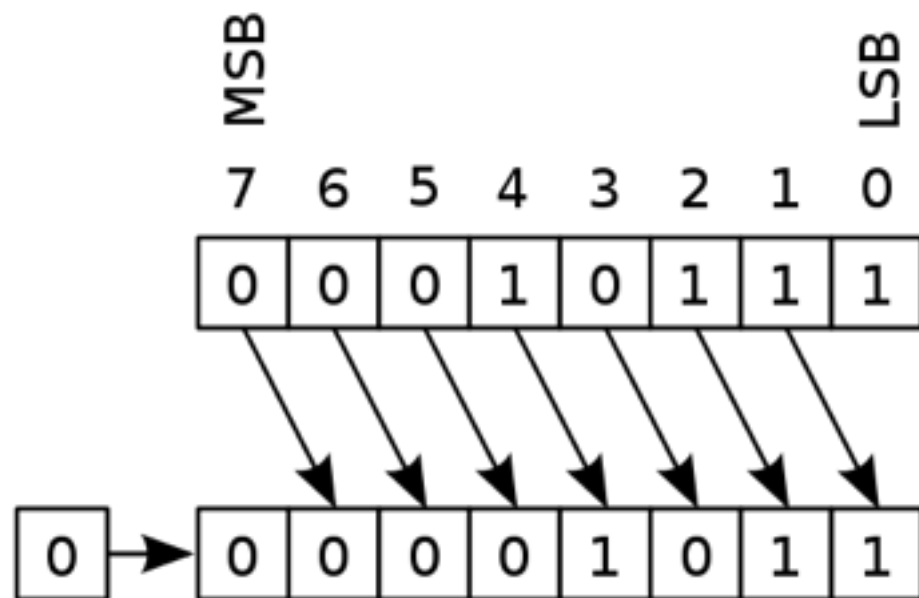


rotate left:

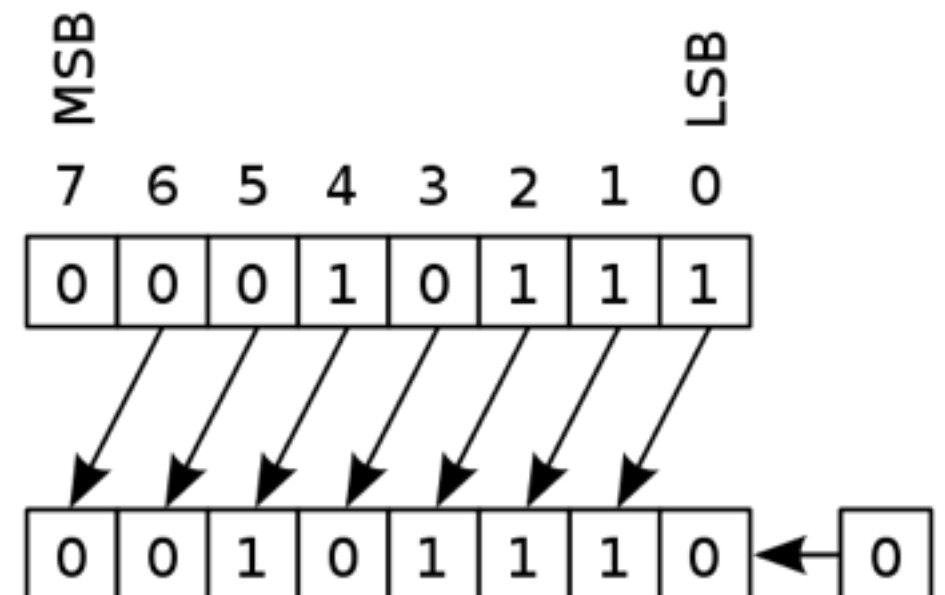


logical shift (like C)

shift right (\gg):



shift left (\ll):



ARM: circular (rotate) shift

shift LSB in:

Rotate Shift Right
ROR

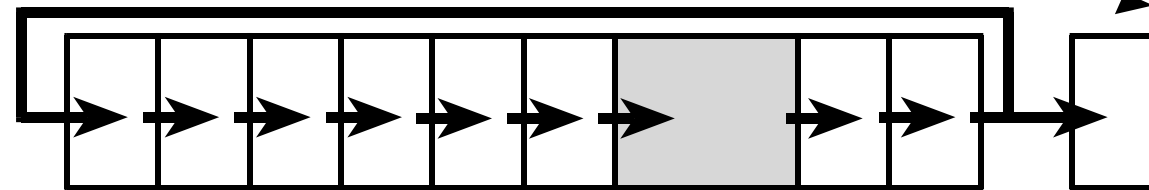
NZ

(C always set)

syntax:

1. `op{s}{cond} {Rd,}Rm, Rs ; Rd=Rm <shift> Rs`
2. `op{s}{cond} {Rd,}Rm, #n ; Rd=Rm <shift> n`

value shifted
out put in C of APSR



shift
amount

contained
in Rs

NOTE: no left rotate

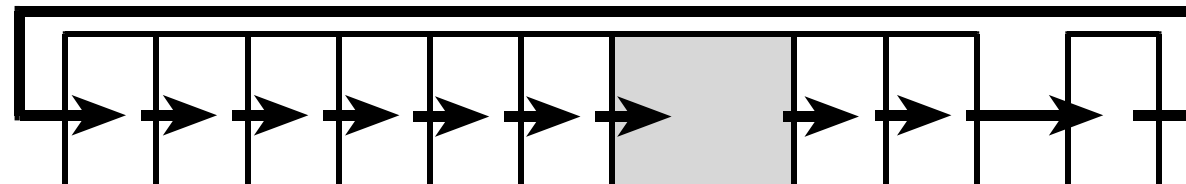
(left rotate of n = right rotate of $32-n$)

ARM: circular (rotate) shift

shift carry in:

Rotate Right Extended
RRX

C of
APSR



n=1

only
shift
by one

NZ

(C always set)

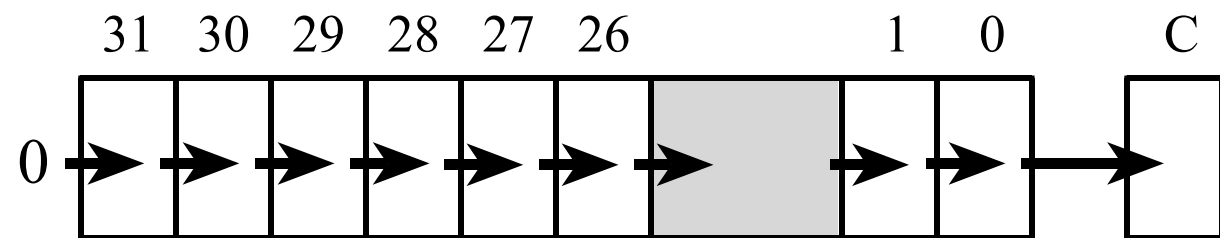
syntax:

`RXX{s}{cond} {Rd,}Rm ; Rd=Rm <shift> 1`

ARM: logical shift

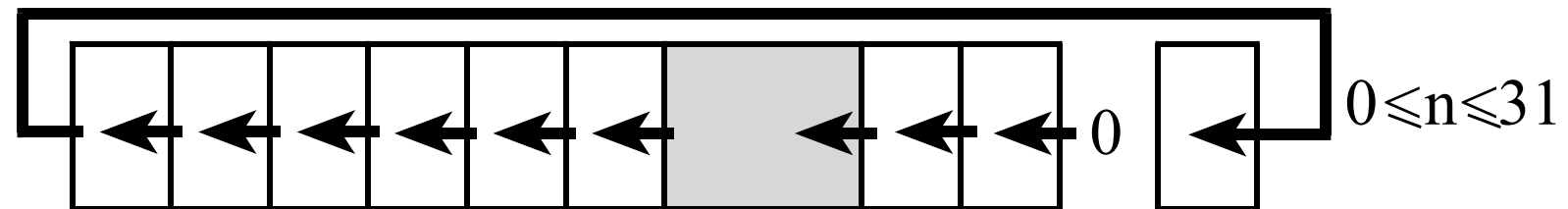
shift 0 in:

Logical Shift Right
LSR



value shifted
out put in C of APSR

Logical Shift Left
LSL



shift
amount

NZ

(C always set)

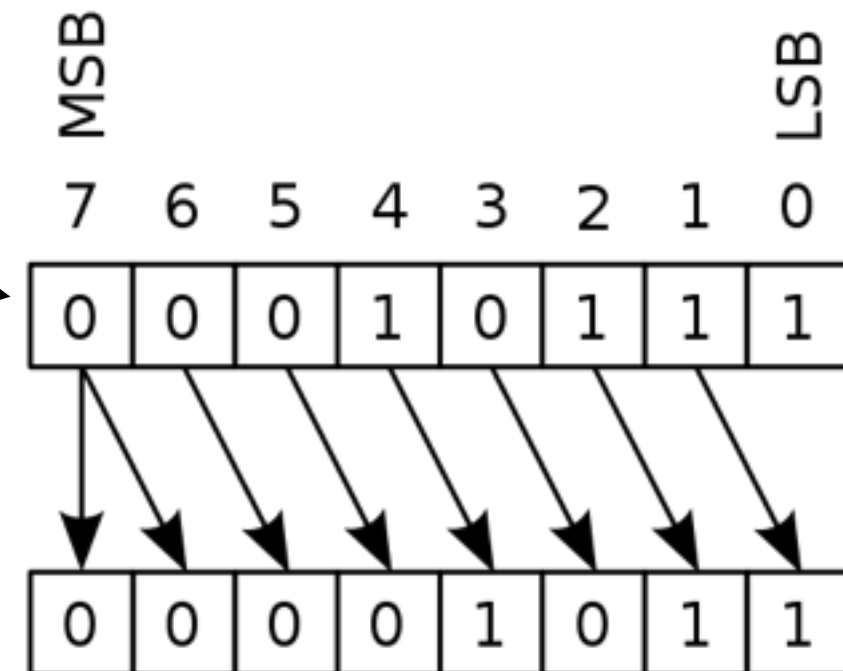
contained
in Rs

syntax:

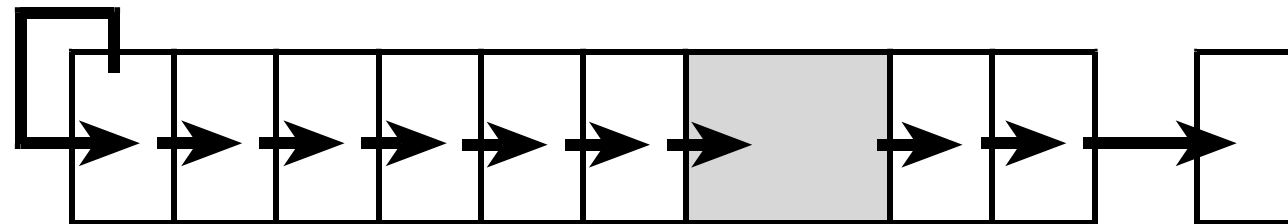
1. `op{s}{cond} {Rd,}Rm, Rs ; Rd=Rm <shift> Rs`
2. `op{s}{cond} {Rd,}Rm, #n ; Rd=Rm <shift> n`

ARM: arithmetic shift (signed)

MSB doesn't
change



Arithmetic Shift Right
ASR



NZ

(C always set)

syntax:

1. `op{s}{cond} Rd,Rm,Rs ;Rd=Rm <shift> Rs`
2. `op{s}{cond} Rd,Rm,#n ;Rd=Rm <shift> n`

sending data: two ways

want to transmit: 1010

```
LDR R1,=PD_DATA_R  
mov R0,#0xA  
str R0,[R1]
```

parallel transmission
(all at once: all pins)

downside: more pins
upside: fast

sending data: two ways

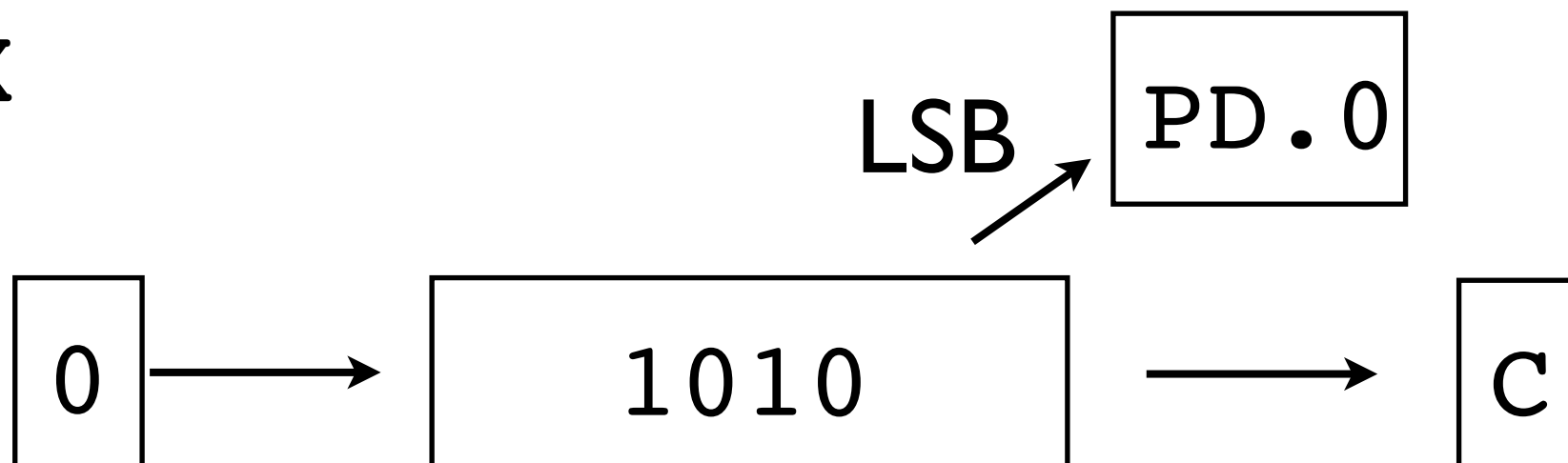
want to transmit: 1010 ← serial transmission
(one bit at a time: one pin)

```
mov R0,#0xa    ;load data
ldr R1,=PD0_DATA_B
mov r2,#4      ;counter
```

downside: slower
upside: fewer pins

STX

```
str R0,[R1]    ;PD0=R0[0]
lsr R0,#1      ;R0>>1
subs R2,#1     ;R2=R2-1
bne STX
```



Timers I

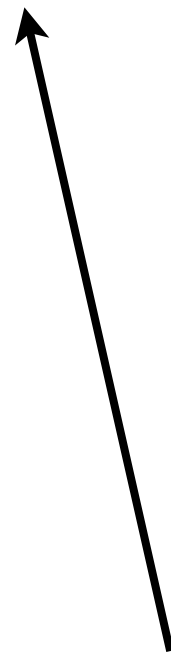
ECE 3710

this is what
gets counted



timer (counter) programming:

1. internal (internal clock)
2. external (events)



will use these
interchangeably

internal:



1. set counter to some initial value, i
2. it counts from i to final value, f
3. rolls over and sets bit, n

```
while(1)
{
    for(cnt=i; cnt>=f; cnt--);

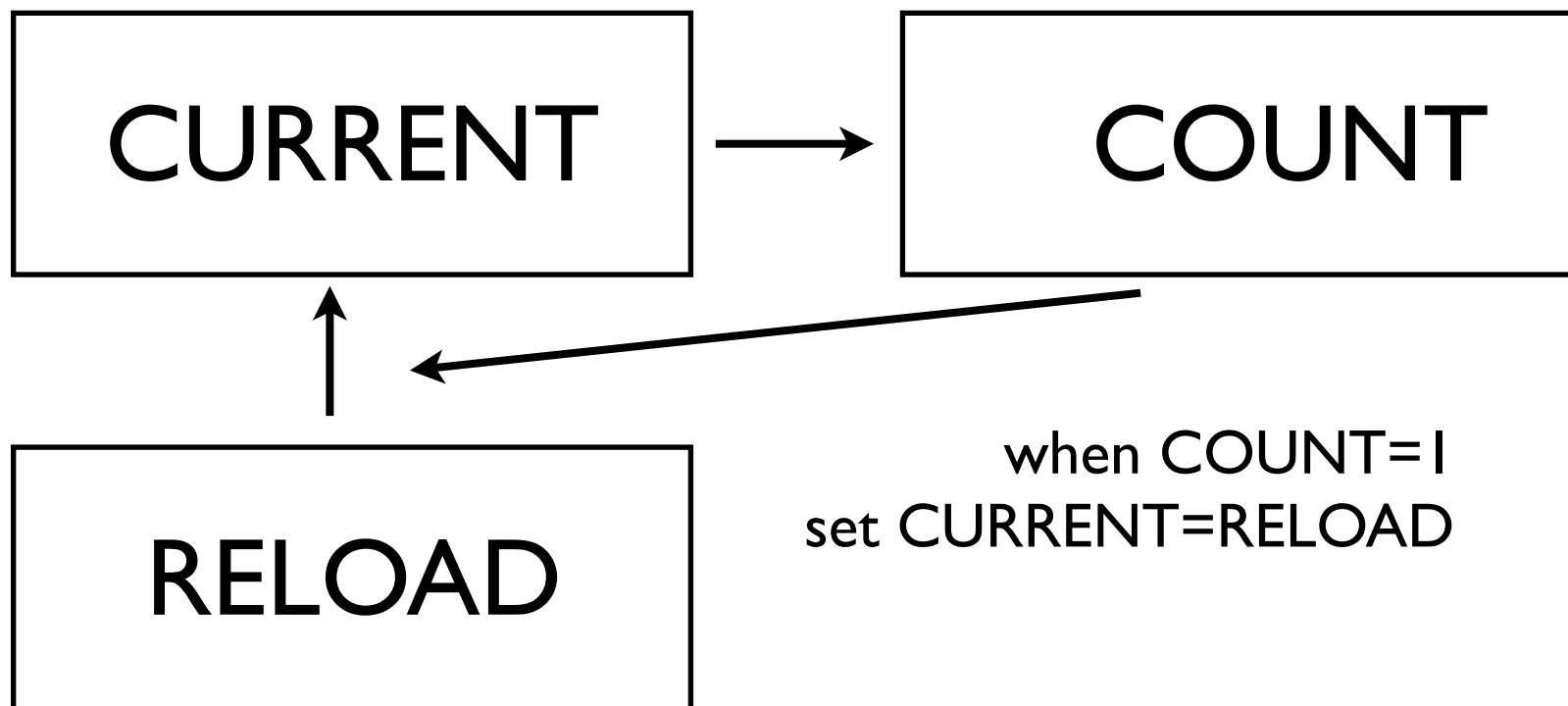
    n = 1;
}
```

note: we can do something
else while uC is counting

SysTick:

1. start at RELOAD
(CURRENT=RELOAD)
2. count down to zero
(CURRENT--)
3. set COUNT=1
4. goto '1.'

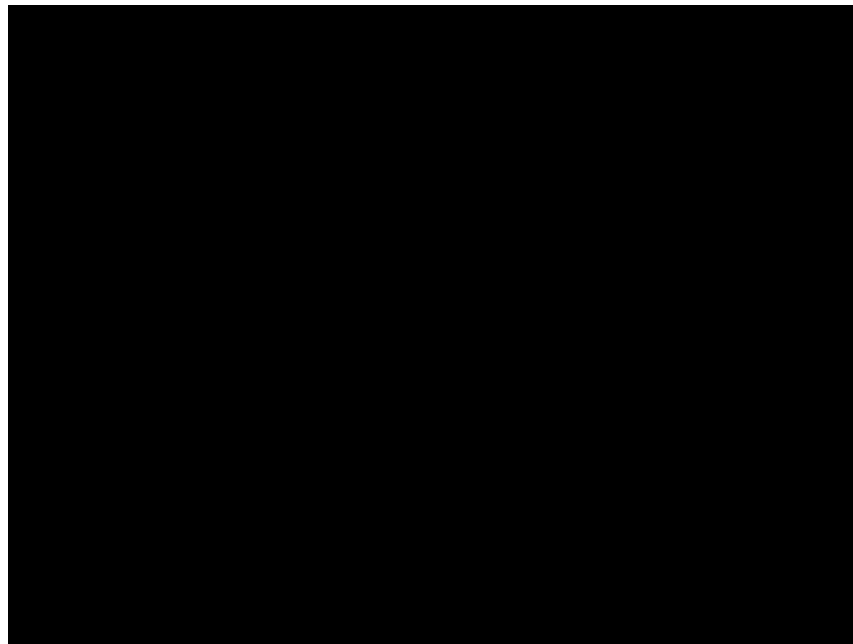
when CURRENT=0
set COUNT=1



when COUNT=1
set CURRENT=RELOAD

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

A couple of things occur to me....



Q:

1. how high can we count?
2. how to set initial value?
3. how to know when count is over?
4. how fast do we count?
5. how much counting can we do?
6. uses? (bah! we only care about theory, right?)

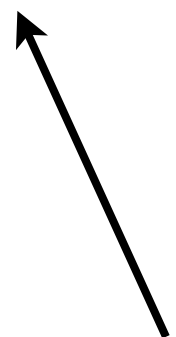
this lecture → *SysTick* Timer

how much and how high?

Cortex-M3: one integrated timer (SysTick)

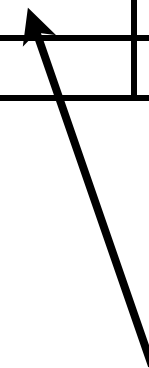
Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

max 24-bits: $0--2^{24}-1$



more or less for peripheral
timer

counts down
from this value
to zero



set initial value

```
ST_RE_R EQU 0xE000E014
ldr R1,=ST_RE_R
mov R0,#0x123456 ;max=0xFFFFFFFF
str R0,[R1]
```

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

starting/stopping SysTick

```
ST_CTRL_R EQU 0xE000E010
ldr R1,=ST_CTRL_R
mov R0,#1           ;start timer
str R0,[R1]
...                ;do stuff
mov R0,#0           ;stop timer
str R0,[R1]
```

Q: above overwrites register is this OK?

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

the rest of ST_CTRL_R

COUNT:

0:=counting

1:=have reached 0

CLK_SRC: counter source

0:=external

1:=internal

INTEN: timer causes interrupt

0:=false

1:=true

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

it counts how fast?

a decrement
every

SysTick: 1/core clock speed

for

CLK_SRC=1

ex: for 8 MHz clock

$$8 \text{ MHz} \rightarrow \frac{1}{8 \times 10^6} = 12.5 \mu\text{s}$$

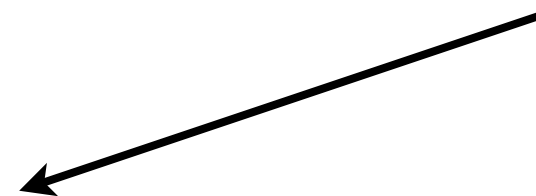
counter still counting?

need to keep
watching it



timer goes from 0x1 to 0x0 → sets COUNT

note: every time we read
ST_CTRL_R
COUNT is cleared




```
ldr R1, =ST_CTRL_R
mov R2, #0x1
WAIT ldr R0, [R1]
     cmp R2, R0, LSR #16 ; R0[16]?=1
     bne WAIT
```

Address	31-24	23-17	16	15-3	2	1	0	Name
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	NVIC_ST_CTRL_R
\$E000E014	0	24-bit RELOAD value						NVIC_ST_RELOAD_R
\$E000E018	0	24-bit CURRENT value of SysTick counter						NVIC_ST_CURRENT_R

it counts how fast?, continued


time to count down:

$$(RELOAD + 1) \times \frac{1}{\text{core clock}}$$


note: setting COUNT takes one cycle

include counting time and time to become aware

in general this will be insignificant w/r/t countdown time



(cmp & b{cond})

Using SysTick:

1. stop timer
2. set initial value
3. clear current value
4. set clock mode
5. enable/disable interrupts
6. start counting

use a single
STR