CEG2400 - Microcomputer Systems

Lecture 5: Hardware initialisation and programming examples

Philip Leong

Last week: Driving TTL from 3.3V

When driving one type of logic from another, need to check voltages and currents are sufficient to do so at the speed that you desire. Level shifters can be used for interfacing.

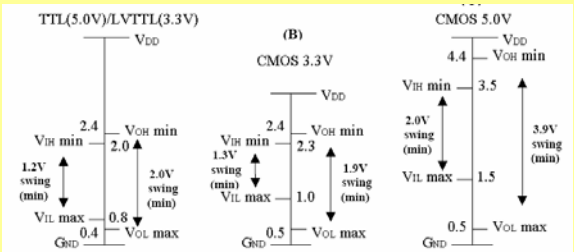


Figure 1. Standard TTL/LVTTL/CMOS/LVCMOS Logic Levels

LPC2131

You should check output current and voltage for output is sufficient to meet the input current and voltage of the connecting device.

V_I	input voltage	pin configured to provide a digital function	[3][4][5] 0	-	5.5	V
V_O	output voltage	output active	0	-	V_{DD}	V
V_{IH}	HIGH-level input voltage		2.0	-	-	V
V_{IL}	LOW-level input voltage		-	-	0.8	V
V_{hys}	hysteresis voltage		-	0.4	-	V
V_{OH}	HIGH-level output voltage	I_{OH} = −4 mA	[6] V_{DD} − 0.4	-	-	V
V_{OL}	LOW-level output voltage	I_{OL} = −4 mA	[6] -	-	0.4	V
I_{OH}	HIGH-level output current	$V_{OH} = V_{DD} - 0.4$ V	[6] −4	-	-	mA
I_{OL}	LOW-level output current	V_{OL} = 0.4 V	[6] 4	-	-	mA
I_{OHS}	HIGH-level short-circuit current	V_{OH} = 0 V	[7] -	-	−45	mA
I_{OLS}	LOW-level short-circuit current	$V_{OL} = V_{DDA}$	[7] -	-	50	mA
I_{pd}	pull-down current	V_I = 5 V	[8] 10	50	150	µA
I_{pu}	pull-up current	V_I = 0 V	[9] −15	−50	−85	µA
		$V_{DD} < V_I < 5$ V	[9] 0	0	0	µA

Example

- One LPC2131 driving two 74LCX244 chips
- LPC2131
  - V_{OL}=0.4V max, V_{OH}=Vdd-0.4=2.9V min
  - I_{OL}=4mA, I_{OH}=-4mA
- 74LCX244
  - V_{IL}=0.8V max, V_{IH}=2V min (ok)
  - I_I=±5uA NB double this for 2 (ok)
- Note for some logic, need to check both I_{IL} and I_{IH}
- Also need to check whether the propagation delay is fast enough

Introduction

- How does the LPC2131 board allow you to download programs?
- How does it allow you to execute your own software?
- What does the J3 jumper do?
- What does the startup code do?

System Control Block

- Provides system features such as
  - External interrupt
  - Crystal oscillator
  - PLL
  - Memory mapping control
  - Power control
  - VPB divider
  - Wakeup timer
- We will discuss those in green today

Reset Circuit

- #RESET goes low when you press SW1
- Upon reset or powerup, ARM executes instruction at address 0x00000000

Actually, should be

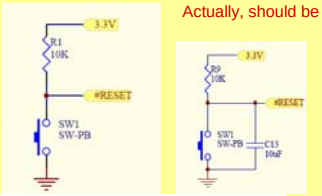


Table 2: ARM exception vector locations

Address	Exception
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt
0x0000 000C	Prefetch Abort (instruction fetch memory fault)
0x0000 0010	Data Abort (data access memory fault)
0x0000 0014	Reserved
	Note: Identified as reserved in ARM documentation, this location is used by the Boot Loader as the Valid User Program key. This is described in detail in "Flash Memory System and Programming" chapter on page 216.
0x0000 0018	IRQ
0x0000 001C	FIQ

Power on

- Wakeup timer ensures everything is stable and ready before allowing instructions to execute
- When using external oscillator, RESET should be asserted for >10ms on powerup
  - What is the RC time constant for the circuit in previous slide?

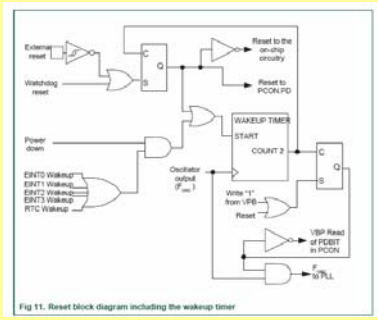


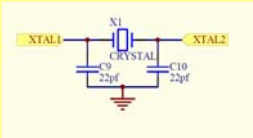
Fig 11. Reset block diagram including the wakeup timer

Reset Code

```
                AREA RESET, CODE, READONLY
                ARM
Vectors         LDR PC, Reset_Addr
                LDR PC, Undef_Addr
                …
; DCD=Define Constant Data (same as DCW)
Reset_Addr      DCD Reset_Handler
                …
                EXPORT Reset_Handler
Reset_Handler
                ; initialise everything here
```


Oscillator

- Crystal + caps connected to XTAL1 and XTAL2 pins to generate a clock
  - A square wave can also be input to XTAL1
- We use 11.0592MHz because it is a multiple of the baud rates we wish to use for the UARTs (11.0592MHz=57600x192)

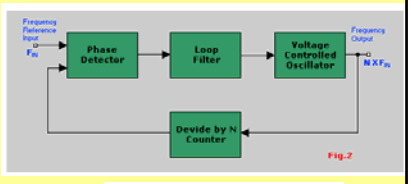


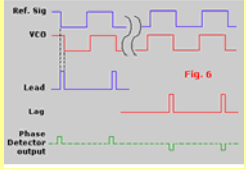
Phase locked loop

- High frequency signals are difficult to handle on a PCB due to the relatively long wires used. This is less of a problem on-chip.
- Sometimes we want a high frequency for high throughput, sometimes we want a low clock frequency for low power and because it generates less electromagnetic interference
- Many chips use a low frequency external clock and multiply on-chip to address these issues
  - A circuit that can do this is a phase locked loop (PLL)

PLL - how it works

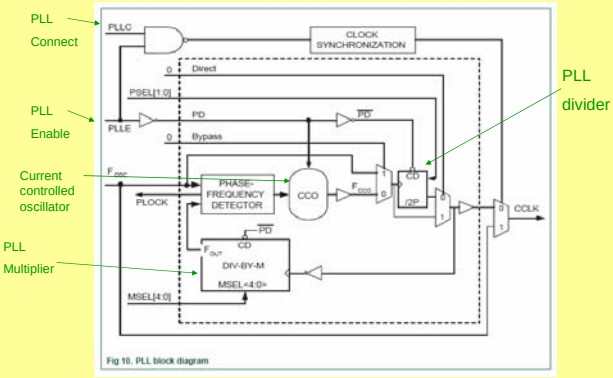
- Feedback system used to multiply clock input clock frequency
- Output of the loop filter sets voltage to VCO so the two frequencies at the phase detector are exactly the same
- Once "locked", Fout=N x Fin





Source: http://www.uoguelph.ca/~antoon/gadgets/pll/pll.html

LPC21xx PLL



PLL

- PLL multiplies the input oscillator frequency F_{OSC} by M to give CCLK
  - CCLK = F_{OSC} x M
- To do this it uses a current controlled oscillator at frequency
  - F_{CCO} = F_{OSC} x M x 2 x P
- Furthermore F_{OSC} must be in range 10-25MHz and F_{CCO} in range 156-320MHz

PLLCFG

3.7.3 PLL Configuration register (PLLCFG - 0xE01F C084)

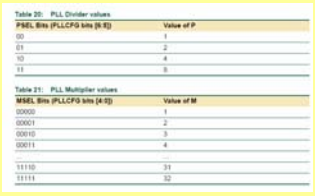
The PLLCFG register contains the PLL multiplier and divider values. Changes to the PLLCFG register do not take effect until a correct PLL feed sequence has been given (see Section 3.7.7 "PLL Feed register (PLLFEED - 0xE01F C08C)" on page 30). Calculations for the PLL frequency, and multiplier and divider values are found in the PLL Frequency Calculation section on page 31.

Table 15: PLL Configuration register (PLLCFG - address 0xE01F C084) bit description

Bit	Symbol	Description	Reset value
4:0	MSEL	PLL Multiplier value. Supplies the value "M" in the PLL frequency calculations. **Note:** For details on selecting the right value for MSEL see Section 3.7.9 "PLL frequency calculation" on page 31.	0
6:5	PSEL	PLL Divider value. Supplies the value "P" in the PLL frequency calculations. **Note:** For details on selecting the right value for PSEL see Section 3.7.9 "PLL frequency calculation" on page 31.	0
7	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

Choosing P and M

- Choose desired F_{OSC} and CCLK (CCLK must be integer multiple)
- Calculate M in range 1-32 (MSEL bits are M-1)
- Calculate P so that F_{CCO} is in correct range (P must be 1,2,4 or 8. PSEL bits are P-1)



Example

- F_{OSC}=10 MHz requires CCLK = 60 MHz
- M = CCLK / Fosc = 60 MHz / 10 MHz = 6.
  - M - 1 = 5 will be written as PLLCFG[4:0]
- Value for P can be derived from P = F_{CCO} / (CCLK x 2), F_{CCO} must be 156-320 MHz. Assuming the lowest allowed frequency for F_{CCO} = 156 MHz, P = 156 MHz / (2 x 60 MHz) = 1.3. The highest F_{CCO} frequency criteria produces P = 2.67. Only solution for P that satisfies both of these requirements and is listed in Table 20 is P = 2. Therefore, PLLCFG[6:5] = 1 will be used.

PLLCON

Table 14: PLL Control register (PLLCON - address 0xE01F C080) bit description

Bit	Symbol	Description	Reset value
0	PLLE	PLL Enable. When one, and after a valid PLL feed, this bit will activate the PLL and allow it to lock to the requested frequency. See PLLSTAT register, Table 16.	0
1	PLLC	PLL Connect. When PLLC and PLLE are both set to one, and after a valid PLL feed, connects the PLL as the clock source for the microcontroller. Otherwise, the oscillator clock is used directly by the microcontroller. See PLLSTAT register, Table 16.	0
7:2	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

PLLFEED

- Incorrect programming of the PLL will cause the uC to operate incorrectly
- The PLL is only updated if a PLLFEED sequence is received
  - Update PLLCFG & PLLCON registers
  - Write 0xAA to PLLFEED
  - Write 0x55 to PLLFEED

PLLSTATUS

Table 16: PLL Status register (PLLSTAT - address 0xE01F C088) bit description

Bit	Symbol	Description	Reset value
4:0	MSEL	Read-back for the PLL Multiplier value. This is the value currently used by the PLL.	0
6:5	PSEL	Read-back for the PLL Divider value. This is the value currently used by the PLL.	0
7	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
8	PLLE	Read-back for the PLL Enable bit. When one, the PLL is currently activated. When zero, the PLL is turned off. This bit is automatically cleared when Power-down mode is activated.	0
9	PLLC	Read-back for the PLL Connect bit. When PLLC and PLLE are both one, the PLL is connected as the clock source for the microcontroller. When either PLLC or PLLE is zero, the PLL is bypassed and the oscillator clock is used directly by the microcontroller. This bit is automatically cleared when Power-down mode is activated.	0
10	PLOCK	Reflects the PLL Lock status. When zero, the PLL is not locked. When one, the PLL is locked onto the requested frequency.	0
15:11	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

PLL Definitions

```
; Phase Locked Loop (PLL) definitions
PLL_BASE       EQU    0xE01FC080     ; PLL Base Address
PLLCON_OFS     EQU    0x00           ; PLL Control Offset
PLLCFG_OFS     EQU    0x04           ; PLL Configuration Offset
PLLSTAT_OFS    EQU    0x08           ; PLL Status Offset
PLLFEED_OFS    EQU    0x0C           ; PLL Feed Offset
PLLCON_PLLE    EQU    (1<<0)         ; PLL Enable
PLLCON_PLLC    EQU    (1<<1)         ; PLL Connect
PLLCFG_MSEL    EQU    (0x1F<<0)      ; PLL Multiplier
PLLCFG_PSEL    EQU    (0x03<<5)      ; PLL Divider
PLLSTAT_PLOCK  EQU    (1<<10)        ; PLL Lock Status
;// <e> PLL Setup
;//   <o1.0..4>   MSEL: PLL Multiplier Selection
;//               <1-32><#-1>
;//               <i> M Value
;//   <o1.5..6>   PSEL: PLL Divider Selection
;//               <0=> 1   <1=> 2   <2=> 4   <3=> 8
;//               <i> P Value
;// </e>
PLL_SETUP      EQU    1
PLLCFG_Val     EQU    0x00000024;   What is M and P? What is F_CCO? CCLK?
```


PLL startup code

```
; Setup PLL
        IF      PLL_SETUP <> 0
        LDR     R0, =PLL_BASE
        MOV     R1, #0xAA
        MOV     R2, #0x55

;  Configure and Enable PLL
        MOV     R3, #PLLCFG_Val     ; 0x24
        STR     R3, [R0, #PLLCFG_OFS]
        MOV     R3, #PLLCON_PLLE
        STR     R3, [R0, #PLLCON_OFS]
        STR     R1, [R0, #PLLFEED_OFS]
        STR     R2, [R0, #PLLFEED_OFS]
```


Wait until ready & switch

```
;  Wait until PLL Locked
PLL_Loop        LDR     R3, [R0, #PLLSTAT_OFS]
                ANDS    R3, R3, #PLLSTAT_PLOCK
                BEQ     PLL_Loop

;  Switch to PLL Clock
                MOV     R3,
   #(PLLCON_PLLE:OR:PLLCON_PLLC)
                STR     R3, [R0, #PLLCON_OFS]
                STR     R1, [R0, #PLLFEED_OFS]
                STR     R2, [R0, #PLLFEED_OFS]
                ENDIF   ; PLL_SETUP
```


VPB Clock

- Processor uses CCLK and peripherals use PCLK (peripherals usually don't run as fast as the processor)
- VPB divider determines relation between them
  - Startup code doesn't change the default of / 4
- What is PCLK in our system?

Table 27: VPB Divider register (VPBDIV - address 0xE01F C100) bit description

Bit	Symbol	Value	Description	Reset value
1:0	VPBDIV	00	VPB bus clock is one fourth of the processor clock.	00
		01	VPB bus clock is the same as the processor clock.	
		10	VPB bus clock is one half of the processor clock.	
		11	Reserved. If this value is written to the VPBDIV register, it has no effect (the previous setting is retained).	
7:2	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

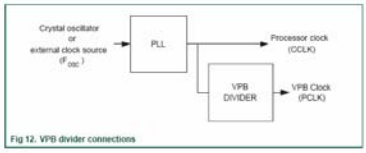


Fig 12. VPB divider connections

Stack space

```
UND_Stack_Size  EQU     0x00000000
SVC_Stack_Size  EQU     0x00000008
ABT_Stack_Size  EQU     0x00000000
FIQ_Stack_Size  EQU     0x00000000
IRQ_Stack_Size  EQU     0x00000080
USR_Stack_Size  EQU     0x00000400

Stack_Size      EQU     (UND_Stack_Size + SVC_Stack_Size +
    ABT_Stack_Size + \
                 FIQ_Stack_Size + IRQ_Stack_Size +
    USR_Stack_Size)

            AREA    STACK, NOINIT, READWRITE, ALIGN=3
Stack_Mem       SPACE   Stack_Size

Stack_Top       EQU     Stack_Mem + Stack_Size
```


Setup stack

```
; Setup Stack for each mode
            LDR     R0, =Stack_Top
;  Enter Undefined Instruction Mode and set its Stack Pointer
            MSR     CPSR_c, #Mode_UND:OR:I_Bit:OR:F_Bit
            MOV     SP, R0
            SUB     R0, R0, #UND_Stack_Size
…
;  Enter User Mode and set its Stack Pointer
            MSR     CPSR_c, #Mode_USR
            MOV     SP, R0
            SUB     SL, SP, #USR_Stack_Size


; Enter the C code
            IMPORT  __main
            LDR     R0, =__main
            BX      R0
```


Memory Mapping Modes

- Wish reset to be flexible
  - download to flash (boot loader)
  - execute our program in flash
  - execute routine in RAM
- Need some way to map different portions of memory to the ARM exception vectors
  - Memory mapping control determines source of this data

Table 2: ARM exception vector locations

Address	Exception
0x0000 0000	Reset
0x0000 0004	Undefined Instruction
0x0000 0008	Software Interrupt
0x0000 000C	Prefetch Abort (instruction fetch memory fault)
0x0000 0010	Data Abort (data access memory fault)
0x0000 0014	Reserved
	Note: Identified as reserved in ARM documentation, this location is used by the Boot Loader as the Valid User Program key. This is described in detail in "Flash Memory System and Programming" chapter on page 216.
0x0000 0018	IRQ
0x0000 001C	FIQ

Memory Mapping modes

Table 3: LPC2131/2/4/6/8 memory mapping modes

Mode	Activation	Usage
Boot Loader mode	Hardware activation by any Reset	The Boot Loader **always** executes after any reset. The Boot Block interrupt vectors are mapped to the bottom of memory to allow handling exceptions and using interrupts during the Boot Loading process.
User Flash mode	Software activation by Boot code	Activated by Boot Loader when a valid User Program Signature is recognized in memory and Boot Loader operation is not forced. Interrupt vectors are not re-mapped and are found in the bottom of the Flash memory.
User RAM mode	Software activation by User program	Activated by a User Program as desired. Interrupt vectors are re-mapped to the bottom of the Static RAM.

Memory mapping modes

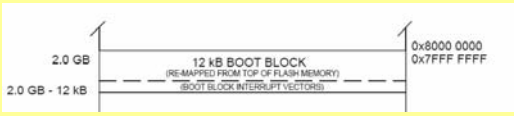
Table 12: Memory Mapping control register (MEMMAP - address 0xE01F C040) bit description

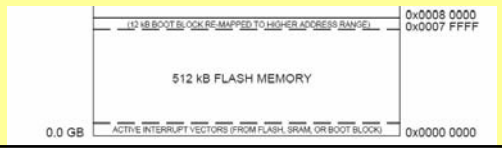
Bit	Symbol	Value	Description	Reset value
1:0	MAP	00	Boot Loader Mode. Interrupt vectors are re-mapped to Boot Block.	00
		01	User Flash Mode. Interrupt vectors are not re-mapped and reside in Flash.	
		10	User RAM Mode. Interrupt vectors are re-mapped to Static RAM.	
		11	Reserved. Do not use this option.	
			Warning: Improper setting of this value may result in incorrect operation of the device.	
7:2	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

So how does the bootloader know to execute your code?

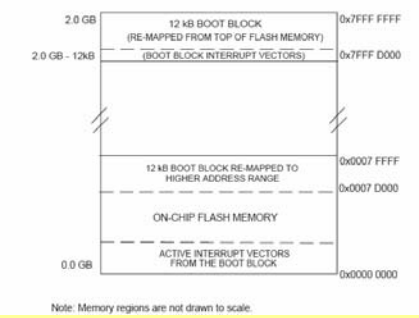
Memory mapping modes

- 12kB boot block remapped to high memory so it is at the same address for devices with different flash sizes





After reset

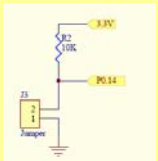


Details

- Remapped area is
  - 32 bytes (size of interrupt buffer area)
  - Additional 32 bytes (to store constants for jumping beyond range of branch instruction)
  - Total 64 bytes
- Same data can be read from both remapped and original locations

Boot loader

- Always runs after reset
- Allows programming of flash memory
  - Low on P0.14 starts the in-system programming command handler (J3 inserted on our board)
  - If high, looks for a valid user program and executes it
  - P0.14 must be pulled high or low by external hardware



Valid user program

- Reserved ARM interrupt vector location (0x0000 0014) should contain the 2's complement of the check-sum of the remaining interrupt vectors.
  - i.e. checksum of all vectors is 0.

Some simple programs

Program 7.3a: add.s — *Add two numbers*

```
1   ; Add two (32-Bit) numbers
2
3           TTL     Ch4Ex3 - add
4           AREA    Program, CODE, READONLY
5           ENTRY
6
7   Main
8           LDR     R1, Value1              ; Load the first number
9           LDR     R2, Value2              ; Load the second number
10          ADD     R1, R1, R2              ; ADD them together into R1 (x = x + y)
11          STR     R1, Result              ; Store the result
12          SWI     &11
13
14  Value1  DCD     &37E3C123               ; First value to be added
15  Value2  DCD     &367402AA               ; Second value to be added
16  Result  DCD     0                       ; Storage for result
17
18          END
```

Following programs from: http://www.arm.com/miscPDFs/9658.pdf
(excellent book on ARM assembly)

64-bit addition

Program 7.7: add64.s — *64 bit addition*

```
1   ; 64 bit addition
2
3           TTL     Ch4Ex8 - add64
4           AREA    Program, CODE, READONLY
5           ENTRY
6
7   Main
8           LDR     R0, =Value1             ; Pointer to first value
9           LDR     R1, [R0]                ; Load first part of value1
10          LDR     R2, [R0, #4]            ; Load lower part of value1
11          LDR     R0, =Value2             ; Pointer to second value
12          LDR     R3, [R0]                ; Load upper part of value2
13          LDR     R4, [R0, #4]            ; Load lower part of value2
14          ADDS    R6, R2, R4              ; Add lower 4 bytes and set carry flag
15          ADC     R5, R1, R3              ; Add upper 4 bytes including carry
16          LDR     R0, =Result             ; Pointer to Result
17          STR     R5, [R0]                ; Store upper part of result
18
19          STR     R6, [R0, #4]            ; Store lower part of result
20          SWI     &11
21
22  Value1  DCD     &12A2E640, &F2100123    ; Value to be added
23  Value2  DCD     &001019BF, &40023F51    ; Value to be added
24  Result  DCD     0                       ; Space to store result
25
26          END
```

Program 7.8: `factorial.s` — *Lookup the factorial from a table by using the address of the memory location*

```
1  ; Lookup the factorial from a table using the address of the memory location
2
3          TTL     Ch4Ex9 - factorial
4          AREA    Program, CODE, READONLY
5          ENTRY
6
7  Main
8          LDR     R0, =DataTable          ; Load the address of the lookup table
9          LDR     R1, Value               ; Offset of value to be looked up
10         MOV     R1, R1, LSL #0x2        ; Data is declared as 32bit - need
11                                         ; to quadruple the offset to point at the
12                                         ; correct memory location
13         ADD     R0, R0, R1              ; R0 now contains memory address to store
14         LDR     R2, [R0]
15         LDR     R3, =Result             ; The address where we want to store the answer
16         STR     R2, [R3]                ; Store the answer
17
18         SWI     &11
19
20         AREA    DataTable, DATA
21
22         DCD     1               ;0! = 1         ; The data table containing the factorials
23         DCD     1               ;1! = 1
24         DCD     2               ;2! = 2
25         DCD     6               ;3! = 6
26         DCD     24              ;4! = 24
27         DCD     120             ;5! = 120
28         DCD     720             ;6! = 720
29         DCD     5040            ;7! = 5040
30 Value   DCB     5
31         ALIGN
32 Result  DCW     0
33
34         END
```


Largest number

Program 9.3: `largest16.s` — *Scan a series of 16 bit numbers to find the largest*

```
1  *       Scan a series of 16 bit numbers to find the largest
2
3          TTL     Ch5Ex5
4          AREA    Program, CODE, READONLY
5          ENTRY
6
7  Main
8          LDR     R0, =Data1              ;load the address of the lookup table
9          EOR     R1, R1, R1              ;clear R1 to store largest
10         LDR     R2, Length              ;init element count
11         CMP     R2, #0
12         BEQ     Done                    ;if table is empty
13 Loop
14         LDR     R3, [R0]                ;get the data
15         CMP     R3, R1                  ;bit is 1
16         BCC     Looptest                ;skip next line if zero
17         MOV     R1, R3                  ;increment -ve number count
18 Looptest
19         ADD     R0, R0, #+4             ;increment pointer
20         SUBS    R2, R2, #0x1            ;decrement count with zero set
21         BNE     Loop                    ;if zero flag is not set, loop
22 Done
23         STR     R1, Result              ;otherwise done - store result
24         SWI     &11
```


Largest number

```
25
26         AREA    Data1, DATA
27
28 Table   DCW     &A152                   ;table of values to be tested
29         ALIGN
30         DCW     &7F61
31         ALIGN
32         DCW     &F123
33         ALIGN
34         DCW     &8000
35         ALIGN
36 TablEnd DCD     0
37
38         AREA    Data2, DATA
39
40 Length  DCW     (TablEnd - Table) / 4   ;because we're having to align
41         ALIGN                           ;gives the loop count
42 Result  DCW     0                       ;storage for result
43
44         END
```