Έγγραφο εξετάσεων MP

Lamprinos Chatziioannou

June 6, 2024

Contents

1 Assembly

Note 1.1

Με την νέα έκδοση υποτίθεται πως δεν επιτρέπονται οι inline functions, παρόλα αυτά, τα χρησιμοποιώ έτσι καθώς αυτό είναι λίγο πολύ και αυτό που δείχθηκε στις διαλέξεις.

```
1 _asm <funcname>(arguments)
2 {
```

```
  // inline assembly
}
```

```
start:
        .fnstart
        // You can only use up to r3 without clearing
        // r4-r8 and r10-r11 need to be saved in the stack
        push {r4,lr}
end:
        // Make sure according to standard to have return at r0
        mov r0, r4
        pop {r4,pc}
        .fnend
```

1.1 List Of Commands

```
ADD R0, R1, #10      ; Adds 10 to R1, result in R0
SUB R2, R3, R4       ; Subtracts R4 from R3, result in R2
MOV R5, #15          ; Moves 15 to R5
MVN R6, #0xFF        ; Moves the bitwise NOT of 0xFF to R6
AND R7, R8, R9       ; Bitwise AND of R8 and R9, result in R7
ORR R10, R11, #1     ; Bitwise OR of R11 and 1, result in R10
EOR R12, R13, R14    ; Bitwise XOR of R13 and R14, result in R12
BIC R15, R0, #0xF0   ; Bitwise AND of R0 with NOT 0xF0, result in R15
CMP R1, #20          ; Compares R1 with 20 (sets condition flags)
CMN R2, #5           ; Compares R2 with negative 5 (sets condition flags)
TST R3, #0x0F        ; Bitwise AND of R3 and 0x0F (sets condition flags)
TEQ R4, R5           ; Bitwise XOR of R4 and R5 (sets condition flags)
LDR R6, [R7, #4]     ; Loads word from memory address (R7 + 4) into R6
LDR r3, =values      ; Also useful, address of variable
STR R8, [R9, #8]     ; Stores word in R8 to memory address (R9 + 8)
LDM R10!, {R1-R4}    ; Loads multiple registers from memory starting at R10
STM R11!, {R5-R7}    ; Stores multiple registers to memory starting at R11
B label              ; Branch to label
BLT label            ; Branch to label if less than
BGT label            ; Branch to label if greater than
PUSH {R0-R3}         ; Pushes registers R0 to R3 onto the stack
POP {R4-R7}          ; Pops registers R4 to R7 off the stack
NOP                  ; No operation
LSL R0, R1, #2       ; Logical shift left R1 by 2, result in R0
ASR R2, R3, #1       ; Arithmetic shift right R3 by 1, result in R2
```

```
ROR R4, R5, #3      ; Rotate right R5 by 3, result in R4
RRX R6, R7          ; Rotate right with extend R7, result in R6
ADC R8, R9, R10     ; Adds R10 + carry to R9, result in R8
SBC R11, R12, R13   ; Subtracts R13 + carry from R12, result in R11
RSB R14, R0, R1     ; Subtracts R0 from R1, result in R14
MLA R2, R3, R4, R5  ; Multiplies R3 by R4, adds R5, result in R2
MUL R6, R7, R8      ; Multiplies R7 by R8, result in R6
UMULL R0, R1, R2, R3; Unsigned multiply R2 by R3, result in R0 and R1
SMULL R4, R5, R6, R7; Signed multiply R6 by R7, result in R4 and R5
LDRB R9, [R10, #1]  ; Loads byte from memory address (R10 + 1) into R9
STRB R11, [R12, #2] ; Stores byte in R11 to memory address (R12 + 2)
SWP R13, R14, [R15] ; Swaps word in R13 with memory at address in [R15]
SWPB R0, R1, [R2]   ; Swaps byte in R0 with memory at address in [R2]

```

2 C

Note 2.1

Κατα κανόνα ο,τι υπάρχει εντός του PDF είναι παρμένο από τους drivers του έτους - μπορεί να χρησιμοποιηθεί αυτούσιο κατα την εξέταση.

2.1 Γενικά περί interrupts

```
__WFI();
__enable_irq(); // Enable interrupts
__disable_irq(); // Disable interrupts
```

2.2 ADC

```

#define R1 (1e6)
#define R2 (1e6)
#define SCALE_FACTOR ((R1+R2)/(R2))
#define VREF (3.3)

int main(void) {
        adc_init(P_ADC);

```

```
        while(1) {
                volatile float vbat;
                volatile int res = (int)adc_read(P_ADC);

                // Scale the adc result to a voltage.
                vbat = (float)res * SCALE_FACTOR * VREF / ADC_MASK;
        }
}
```

2.3 Platform

- ☐ Add buttons

2.4 Queue

- Holding only type int

```
#define QUEUE_SIZE 128
void queue_init(Queue *queue, uint32_t size);
int queue_enqueue(Queue *queue, int item);
int queue_dequeue(Queue *queue, int *item, int *halfP) ;
int queue_is_full(Queue *queue);
int queue_is_empty(Queue *queue);
```

2.5 Delay

```
void delay_ms(unsigned int ms);
void delay_us(unsigned int us);
void delay_cycles(unsigned int cycles);
```

2.6 UART

Θεωρούμε την σειριακή επικοινωνία, όπου ζητείται, UART.

```
// usually 115200
void uart_init(uint32_t baud);
// Enables UART transmission and reception.
void uart_enable(void);
// Transmit a single character.
void uart_tx(uint8_t c);
```

```
// Set the UART receive callback function
void uart_set_rx_callback(uart_rx_isr);
void uart_print(char *str);
```

Useful snippet

```
Queue rx_queue; // Queue for storing received characters

// Interrupt Service Routine for UART receive
void uart_rx_isr(uint8_t rx) {
        // Check if the received character is a printable ASCII character
        if (rx >= 0x0 && rx <= 0x7F ) {
                // Store the received character
                queue_enqueue(&rx_queue, rx);
        }
}
```

2.7 Timer/Counter

Note 2.2

Γίνεται η θεώρηση, όπως και στις διαλέξεις πως είναι count-down timer: θέτεις maxvalue, βασει του τύπου που φαίνεται παρακάτω και με την συχνότητα του ρολογιού μειώνεται μέχρι να φτάσει το 0 οπότε και θα στείλει interrupt.

`maxval = round(T * Freq)`, where T is the interrupt period, Freq the clock frequency: useful `CLK_FREQ/TIMESPERSECOND`, the first one is a macro:

```
// Interrupt 1000 times per second
CLK_FREQ/1000
```

```
void timer_init(CLK_FREQ/Y);
void timer_set_callback(my_isr);
void timer_enable();
void timer_disable();
```

2.8 GPIO

2.8.1 PINS

Some interesting *pins* are:

- P_{SW}, P_{LED1}, P_{LED2}

2.8.2 Usage

- PinModes: Reset/Input/Output/PullDown/PullUp
- TriggerModes: None,Rising,Falling

```
void gpio_set_mode(Pin PIN, );
// Output
void gpio_set(Pin PIN, int value);
void gpio_toggle(Pin PIN):
// Input
int gpio_get(Pin PIN);
void gpio_set_trigger(Pin pin, TriggerMode trig);
void gpio_set_callback(Pin pin, void (*callback)(int status));
```

```
/*! \brief Sets a range of sequential pins to the specified value.
 *  \param pin_base  Starting pin.
 *  \param count     Number of pins to set.
 *  \param value     New value of the pins.
 */
void gpio_set_range(Pin pin_base, int count, int value);

/*! \brief Returns the value of a range of sequential pins.
 *  \param pin_base  Starting pin.
 *  \param count     Number of pins to set.
 *  \returns         Value of the pins.
 */
unsigned int gpio_get_range(Pin pin_base, int count);
```

$

2.9 PWM

```
#define PWM_PERIOD 1000   // PWM period in microseconds
#define PWM_PIN PA_10 // Set PWM PIN

// PWM function to set duty cycle
// duty cycle is percentage of PWM_PERIOD:
// if "active" for 50% of the time -> duty_cycle=50
void pwm_perform__cycle(Pin pin, uint8_t duty_cycle) {
    // Calculate the pulse width based on duty cycle
```

```
    uint32_t pulse_width = (duty_cycle * PWM_PERIOD) / 100;

    // Set the GPIO pin high for pulse_width microseconds
    gpio_set(pin, HIGH);
    delay_us(pulse_width);

    // Set the GPIO pin low for (PWM_PERIOD - pulse_width) microseconds
    gpio_set(pin, LOW);
    delay_us(PWM_PERIOD - pulse_width);
}

int main() {
    // Example usage
    gpio_set_mode(PWM_PIN, Output);
    pwm_init(PWM_PIN);
    pwm_set_duty_cycle(PWM_PIN, 75); // 75 duty cycle
}

```

2.10 Extra Long timer

Sadly this does not use `timer.h`, but, since it is such low level, I am pretty sure it will be accepted in that scenario. It was only created due to a previous exam task asking for 10minute interrupts. That can not happen with builtin memory.

```
// Does not conflict with given timer (timer.h uses SysTick)0
// We use that to gain advantage of the prescaler.

// Assuming a 10MHz clock source, and a 1:10000 prescaler to get a 1kHz tick rate
// Adjust these values according to your microcontroller's clock configuration
// Equation is
#define TIM0CLKFREQ 10000000
TIM0->CR1 |= TIM_CR1_URS; // Only overflow generates an interrupt
TIM0->PSC = 9999; // Prescaler value
TIM0->ARR = round(TIM0CLKFREQ/PSC); // Auto-reload value for 10 minutes at 1kHz tick rate
TIM0->DIER |= TIM_DIER_UIE; // Enable update interrupt

// Enable Timer0 interrupt in NVIC
NVIC_EnableIRQ(TIM0_IRQn);

// Start Timer0
TIM0->CR1 |= TIM_CR1_CEN; // Enable timer counter
```

$$ARR = round(T * Freq/PSC) \quad (1)$$

όπου:

- T: περίοδος σε δευτερόλεπτα
- Freq: συχνότητα ρολογιού σε HZ
- PSC: κλίμακα prescaler

```
// In the ARM Cortex-M architecture, the names of interrupt service
// routines are standardized. They are named using a convention that
// includes the peripheral name followed by _IRQHandler. For example,
// for Timer0, the convention is TIM0_IRQHandler.
void TIM0_IRQHandler(void)
{
    // The code
}
```