EECS 373 Midterm 1
Fall 2017

27 September 2017

No calculators or reference material.

Pledge: I have neither given nor received aid on this exam nor observed anyone else doing so.

Signature:

Name:

Unique name:

1. **(10 pts.)** Build process

   (a) **(3 pts.)** Consider the following makefile definition and rule.

   ```
   CFLAGS = -mcpu=cortex-m3 -mthumb -g3 -O1

   %.o: %.c Makefile
     ${PREFIX}gcc ${CFLAGS} -c -o $@ $<
     ${PREFIX}objdump -D $@ > `basename $@ .o`.lst
     ${PREFIX}nm $@ > `basename $@ .o`.nm
   ```

   If the user has just completed a build, changes the CFLAGS optimization level in the makefile to 3, and types make again, will the source files be recompiled? **Yes** / ~~**No**~~

   (b) **(4 pts.)** Sometimes a linker adds new labels and code to a program. Here is one such example.

   ```
   __*ABS*0x4000001_veneer:
   ldr     pc, [pc, #-4]   ; 2000001c <__*ABS*0x4000001_veneer+0x4>
   streq   r0, [r0], #-1
   ```

   When a program branches to this label, will the streq instruction be executed? ~~**Yes**~~ / **No**

   (c) **(3 pts.)** Circle the tool with the job of finalizing the offset literals within branch instructions.

   - ~~Compiler~~
   - ~~Assembler~~
   - Linker

- ~~Make~~

2. **(10 pts.)** Interrupts

   (a) **(2 pts.)** Using one sentence, explain preemption.
   Preemption is the process of temporarily interrupting a running task to run another.

   (b) **(1 pts.)** Using one sentence, explain why it is useful in embedded systems. **Preemption allows for system critical tasks to take priority when necessary.**

   (c) **(7 pts.)** You have designed a robot controller with three external interrupts.
   - Interrupt A occurs when the robot detects it will soon collide with an object. It is in the highest priority group (priority group 1) and has a subpriority of 2.
   - Interrupt B occurs when the robot's battery energy is low. It is also in the highest priority group (priority group 1) and has a subpriority of 3.
   - Interrupt C occurs when the robot enters a power saving mode. It is in priority group 2 and has a subpriority of 1.

   For each of the following situations, indicate the order in which interrupts are handled by the processor, using "A", "B", and "C". **Assuming "handle" means "enter ISR".**

   i. All of the interrupts occur at the same time. **A, B, C**
   ii. Interrupt C is executing when interrupts A and B trigger at the same time. **A, B, C**
   iii. Interrupt B is executing when interrupts A and C trigger at the same time **B, A, C**

3. **(10 pts.)** MMIO
Write a C function that takes an integer input, compares the input to a threshold, and turns on a LED if and only if the input is greater than the threshold. The LED is mapped to memory address 0x45001234 and has two states: on and off. The threshold value is an integer with range [0:255] and can be read from the register mapped to memory address 0x4500FFFF. The LED is active-high.

```
void LEDoutput(int val) {
  volatile uint32_t * threshold_reg = (uint32_t *)(0x4500ffff);
  volatile uint8_t * led_reg = (uint8_t *)(0x45001234);

  if(val > *threshold_reg) {
    *led_reg = 1;
  } else {
    *led_reg = 0;
  }
}
```

4. **(10 pts.)** The C implementation of greatest_arr is on this page. The next page contains an assembly version with some lines missing. Fill in the missing lines so that the function works properly and is ABI compliant. The *fill* function takes in an array and places some values into it.

   This is a simple example of allocating an integer on the stack. Note that the stack grows downwards, and the integer is deallocated by adding to the stack pointer.

```
allocate_int:
    sub sp, #4
    mov r0, #2
    str r0, [sp]
    add sp, #4
```

```
void greatest_arr(uint32_t arr1[], uint32_t return_arr[]){
    uint32_t arr2[5];
    int i = 0;

    fill(arr2);
    for(; i < 5; i++){

        if(arr1[i] >= arr2[i]){
            return_arr[i] = arr1[i];
        } else {
            return_arr[i] = arr2[i];
        }
    }
    return;
}
```

```
greatest_arr:
  push{r4, r5, r6, lr}
  sub sp, #20
  mov r2, r0     //r2 = arr1
  mov r0, sp     // r0 = arr2 for fill, r1 = return_arr
  push{r0, r1, r2}
  bl fill
  pop{r0, r1, r2}
  mov r3, #0    // r3 = i
  mov r4, #5    // end loop
loop:
  lsl r0, r3, #2    // r0 = i*4
  ldr r5, [r2,r0]  // r5= arr1[i]
  ldr r6, [sp, r0] // r6 = arr2[i]
  cmp r5, r6
  blt else
if:
  str r5, [r1,r0]    //return_arr[i] = arr1[i]
  b endif
else:
  str r6, [r1, r0]    //return_arr[i] = arr2[i]
endif:
  add r3, r3, #1 // i= i+ 1
  cmp r3, r4
  blt loop    // branch if i < 5
  add sp, #20
  pop{r4, r5, r6, lr}
  bx lr
```

5. **(10 pts.)** Instruction Encoding
Provide the machine code in hexadecimal for the following instructions. See the attached reference. Assume UAL.

   (a) **(3 pts.)** strb r1, [r2, #7]: **71d1**

   (b) **(4 pts.)** ldrh r7, [r2, #100]: **f8b2 7064**

   (c) **(3 pts.)** cmp r2, #32: **2a20**

6. **(10 pts.)** Pointer Exercise
Provide the values of the memory locations after this code executes. Assume the initial memory values are zero. Express memory contents in HEX. Accesses are little endian.

```
mov r0, #100
mov r1, 0x12
movt r1, 0xef
movw r2, 0xabcd
str r2, [r0],0
str r1, [r0,2]!
strh r1,[r0,-1]
strb r2,[r0]
```

The following list of addresses are in decimal.

- 100: CD
- 101: 12
- 102: CD
- 103: 00
- 104: EF
- 105: 00
- 106: 00
- 107: 00
- 108: 00

7. **(10 pts.)** Verilog
Create a Verilog module that generates a pulse-width modulated signal. That means the signal will be a square wave for which the amount of time within each period spent high is controlled (modulated).

This module is controlled by writing to two memory-mapped input-output registers. A register at memory address 0x40050000 sets the period of the pulse-width modulated signal (clock ticks per period) and a register at address 0x40050004 controls the pulse width of the pwm signal (clock ticks per pulse).

You can assume that any period or duty cycle values this module receives will be valid. In other words, you do not need to verify that the duty cycle value is lower than the period value.

APB bus read and write timing diagrams are provided on the last page for your reference.

```
module generate_pwm (
  input PCLK,
  input PRESERN,
```

```
  input PSEL,
  input PENABLE,
  input [7:0] PADDR,
  output PREADY,
  output PSLVERR,
  input PWRITE,
  input [31:0] PWDATA,
  output [31:0] PRDATA,
  output pwm
)
  reg [31:0] count, width, period;
  reg pwm;

  assign PREADY = 1;
  assign PSLVERR = 0;

  always@(posedge PCLK)
  begin

    if (PSEL && PENABLE && PWRITE && !PADDR[2]
      period <= PWDATA;
    else if (PSEL && PENABLE && PWRITE && PADDR[2]
      width <= PWDATA;

    if(count >= period)
      count <= 32b0;
    else
      count <= count + 1;

    if (count < width)
      pwm <= 1;
    else
      pwm <= 0;
    end
endmodule
```