

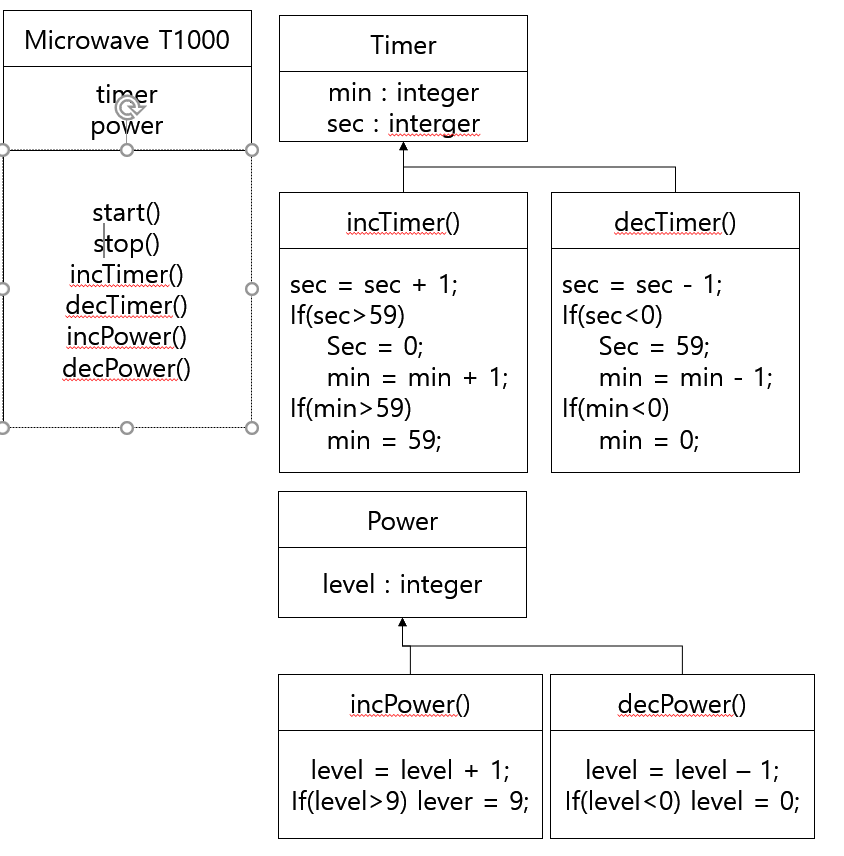
**Fall Semester 2021**

**CSE467M: Embedded Computing Systems**

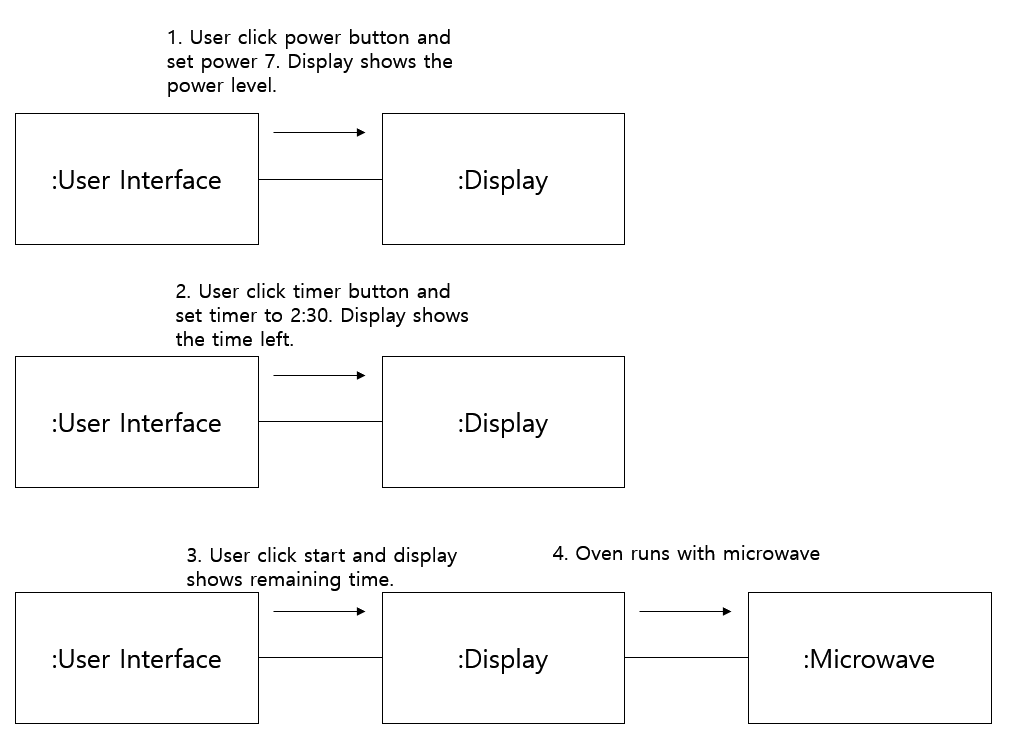
**Homework #1**

**Chapter 1 problems (textbook pages 52 and 53):**

1. **Q1-1 (10 points)**
   1. The requirement is the user's explanation of what the finished product should do. But the specification is more precise and it serves as contract between the customer and the architects.
2. **Q1-7 (10 points)**
   1. Describing how the system implements those functions is the purpose of the architecture. The specification does not say how the system does things, only what the system does.
3. **Q1-8 (10 points)**
   1. Architecture design step.
4. **Q1-9 (10 points)**
   1. Architecture design step.
5. **Q1-11 (10 points)**
   1. Both ‘Components’ phase and ‘System integration’ phase.
6. **Q1-12 (10 points)**
   1. Both of approaches are to design an embedded system. Top down approach begins with most abstract description of the system while bottom-up approach start with components to build a system.
7. **Q1-13 (10 points)**
   1. Defining a large and complex embedded system is theexample of design problem that is best solved using top-down techniques.
8. **Q1-14 (10 points)**
   1. System’s performance and memory capacity is the example of design problem that is best solved using bottom-up techniques.
9. **Q1-23 (10 points)**

****

**10) Q1- 24 (10 points)**

****

**Chapter 2 problems:**

1. **Q2-1 (10 points)**
   1. Little-endian : the lowest-order byte residing in the low-order bits of the word
   2. Big-endian : the lowest-order byte stored in the highest bits of the word
2. **Q2-2 (10 points)**
   1. Von Neumann architectures: A computer whose memory holds both data and instructions
   2. Harvard architectures: It has separate memories for data and program.
3. **Q2-10 (10 points)**
   1. Yes
4. **Q2-11 (10 points)**
   1. 8 levels
5. **Q2-14 (10 points)**
   1. The C55x has four 40-bit accumulators AC0,AC1,AC2 and AC3.
6. **Q2-25 (10 points)**
   1. Fetch packets are instructions that are fetched in groups whereas Execute packet is a set of instructions that execute together.
7. **Implement C statement x = a + b – c\*(d + e); using ARM instruction set.**

**Please comment all instructions in your assembly code. (40 points)**

**\*\* I will use a frame pointer to hold the variables: a is at -24, b at -28, c at -32, d at -36, e at -40 and x at -44**

LDR r0, [fp, #-24] ; load value a into r0

LDR r1, [fp, #-28] ; load value b into r1

ADD r0, r0, r1 ; a + b, add two value and store it in r0

LDR r1, [fp, #-36] ; load value d into r1

LDR r2, [fp, #-40] ; load value e into r2

ADD r1, r1, r2 ; d + e, add two value and store it in r1

LDR r2, [fp, #-32] ; load value c into r2

MUL r1, r1, r2 ; c\*(d + e), multiply two value and store it in r1

SUB r0, r0, r1 ; a+b – c\*(d+e) and then store it r0

STR r0, [fp, #-44] ; save result value into x

1. **Implement the following C if statement using ARM instruction set.**

**if (a<b) {**

**x = c – d;**

**y = 10\*(e + f);**

**}**

**else {**

**x = 5\*c + d;**

**y = e – f;**

**}**

**Please comment all instructions in your assembly code. (40 points)**

**\*\* I will use a frame pointer to hold the variables: a is at -24, b at -28, c at -32, d at -36, e at -40, f at -44, x at -48 and y at -52**

LDR r0, [fp, #-24] ; load value a into r0

LDR r1, [fp, #-28] ; load value b into r1

CMP r0, r1 ; compare a, b

BLT .L2

.L2 : LDR r0, [fp, #-32] ; load value c into r0

LDR r1, [fp, #-36] ; load value d into r1

SUB r0, r0, r1 ; subtract c – d

STR r0, [fp, #-48] ; save result value into x

LDR r1, [fp, #-40] ; load value e into r1

LDR r2, [fp, #-44] ; load value f into r2

ADD r1, r1, r2 ; e+f, add two value and store it in r1

MOV r2, #10 ; load 10 into register r2

MUL r1, r2, r1 ; 10\*(e + f), multiply two value and store it in r1

STR r1, [fp, #-52] ; save result value into y

B .L3

MOV r0, #5 ; load 5 into register r0

LDR r1, [fp, #-32] ; load value c into r1

MUL r0, r0, r1 ; 5\*c, multiply two value and store it in r0

LDR r1, [fp, #-36] ; load value d into r0

ADD r0, r0, r1 ;(5\*c) + d, add two value and store it in r0

STR r0, [fp, #-48] ; save result value into x

LDR r0, [fp, #-40] ; load value e into r0

LDR r1, [fp, #-44] ; load value f into r1

SUB r0, r0, r1 ; subtract e – f

STR r0, [fp, #-52] ; save result value into y

.L3 :

1. **Implement the following C for statement using ARM instruction set.**

**for (i=0, f=0; i<5; i++)**

**f = f + (a[i] + b[i])\*x[i] – y[i];**

**Please comment all instructions in your assembly code. (80 points)**

**\*\* I will use a frame pointer to hold the variables:**

**i is at -24, f at -28, a at -40, b at -60, x at -80 and y at -100**

.INIT:

mov r3, #0 ; set r3 to 0

str r3, [fp, # -24] ; store 0 into i

mov r3, #0 ; set r3 to 0

str r3, [fp, # -28] ; store 0 into f

.COMPARE:

ldr r3, [fp, # -24] ; set r3 to value of i

cmp r3, #5 ; compare i with number 5

blt .LOOP ; jump to :LOOP if r3 is less than number 5

b .OUT ; jump to :OUT

.LOOP:

ldr r3, [fp, # -24] ; set r3 to value of i

mov r3, r3, asl #2 ; set r3 to value of i \* 4

sub r3, fp, r3 ; set r3 to i-th element offset from frame pointer

sub r0, r3, #40 ; set r0 to an address of a array’s i-th element

ldr r3, [fp, # -24] ; set r3 to value of i

mov r3, r3, asl #2 ; set r3 to value of i \* 4

sub r3, fp, r3 ; set r3 to i-th element offset from frame pointer

sub r1, r3, #60 ; set r1 to an address of b array’s i-th element

ldr r2, [r0, #0] ; set r2 to value of a[i]

ldr r3, [r1, #0] ; set r3 to value of b[i]

add r0, r2, r3 ; set r0 to value of a[i] + b[i]

ldr r3, [fp, # -24] ; set r3 to value of i

mov r3, r3, asl #2 ; set r3 to value of i \* 4

sub r3, fp, r3 ; set r3 to i-th element offset from frame pointer

sub r2, r3, #80 ; set r2 to an address of x array’s i-th element

ldr r1, [r2, #0] ; set r1 to value of x[i]

mul r0, r0, r1 ; set r0 to value of (a[i] + b[i])\*x[i]

ldr r3, [fp, # -24] ; set r3 to value of i

mov r3, r3, asl #2 ; set r3 to value of i \* 4

sub r3, fp, r3 ; set r3 to i-th element offset from frame pointer

sub r2, r3, #100 ; set r2 to an address of y array’s i-th element

ldr r1, [r2, #0] ; set r1 to value of y[i]

sub r0, r0, r1 ; set r0 to value of (a[i] + b[i])\*x[i] – y[i]

ldr r3, [fp, # -28] ; set r3 to value of f

add r0, r0, r3 ; set r0 to value of f + (a[i] + b[i])\*x[i] – y[i]

str r0, [fp, # -28] ; f = f + (a[i] + b[i])\*x[i] – y[i]

ldr r3, [fp, # -24] ; set r3 to value of i

add r3, r3, #1 ; i + 1

str r3, [fp, # -24] ; i = i+1

b .COMPARE

.OUT:

**10) Implement the following C for statement using Microchip PIC32 FIR Filter example.**

**for (i=0, f=0; i<10; i++)**

**f = f + (a[i]\*b[i] - c[i])\*x[i] + y[i];**

**Please comment all instructions in your assembly code. (80 points)**

**\*\* I assume that**

i variable : 0 byte offset from fp

f variable : 4 bytes offset from fp

a array : 8 bytes offset from fp

b array : 48 bytes offset from fp

c array : 88 bytes offset from fp

x array : 128 bytes offset from fp

y array : 168 bytes offset from fp

.L2 :

lw $2, 0($fp) ; load i value into $2

slt $2, $2, 10 ; compare i and number 10

beq $2, $0, .L3 ; i equals 10 then jump to L3

nop

lw $2, 0($fp) ; load i value into $2

sll $2, $2, 2 ; shift 2 bits to get the offset of next element

addu $2, $2, $fp ; add offset with fp

lw $3, 8($2) ; load a[i] into $3

lw $2, 0($fp) ; load i value into $2

sll $2, $2, 2 ; shift 2 bits to get the offset of next element

addu $2, $2, $fp ; add offset with fp

lw $2, 48($2) ; load b[i] into $2

mul $3, $3, $2 ; a[i] \* b[i]

lw $2, 0($fp) ; load i value into $2

sll $2, $2, 2 ; shift 2 bits to get the offset of next element

addu $2, $2, $fp ; add offset with fp

lw $2, 88($2) ; load c[i] into $2

sub $3, $3, $2 ; a[i] \* b[i] - c[i]

lw $2, 0($fp) ; load i value into $2

sll $2, $2, 2 ; shift 2 bits to get the offset of next element

addu $2, $2, $fp ; add offset with fp

lw $2, 128($2) ; load x[i] into $2

mul $3, $3, $2 ; (a[i] \* b[i] - c[i])\*x[i]

lw $2, 0($fp) ; load i value into $2

sll $2, $2, 2 ; shift 2 bits to get the offset of next element

addu $2, $2, $fp ; add offset with fp

lw $2, 168($2) ; load y[i] into $2

add $3, $3, $2 ; (a[i] \* b[i] - c[i])\*x[i] + y[i]

lw $2, 4($fp) ; load f into $2

add $3, $3, $2 ; f + (a[i] \* b[i] - c[i])\*x[i] + y[i]

sw $3, 4($fp) ; store result

lw $2, 0($fp) ; load value i

addiu $2, $2, 1 ; increment loop count

sw $2, 0($fp) ; store loop count

b .L2 ; jump to L2

nop

.L3: