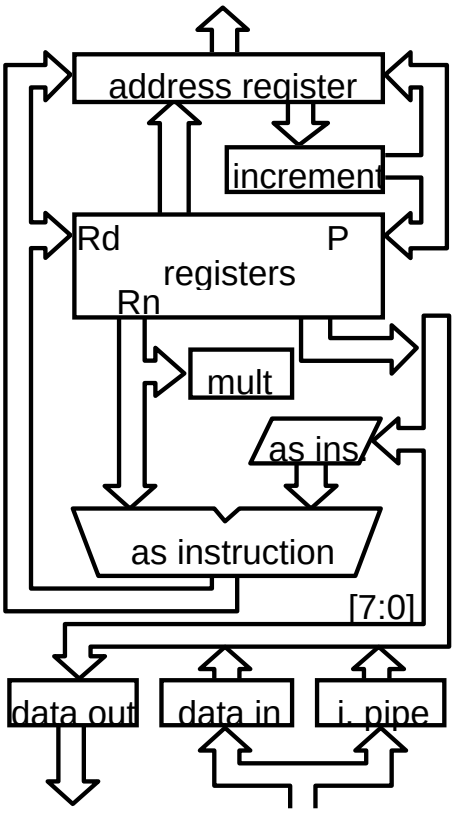
National Sun Yat-Sen University
ASSEMBLY LANGUAGE AND MICROCOMPUTER
Final Exam
1:15-3:15 PM Jan 13 2011

Name: _____

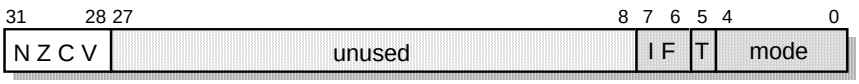
Note: Although there are more than 100 points for this exam, the maximum score you can get is 100 points.

1. Refer to the following 3-stage (fetch, decode, execute) ARM7 pipeline data path. ***(14 pts)***
   (a) Find out the number of cycles it will takes to run the ARM instruction ***ADD r0, r1, r2 LSL #12*** at the execution stages. ***(3 pts)***
   (b) Show the datapath activity at each cycle. ***(5 pts)***
   (c) Fill the following immediate field of the instruction used to return from the undefined instruction trap. You have to explain the reason. ***(6 pts)***

   SUBS pc, r14, _____



2. Write a short ARM code to set the N flag in CPSR to 1. ***(5 pts)***



3. Answer the following short questions: ***(9 pts)***
   (a) Explain what "**caller-saved**" register variables mean, and list these registers defined in APCS (ARM Procedure Call Standard) . ***(4 pts)***
   (b) Describe how to use SWP instruction in a program to realize the check-and-lock operation of a binary semaphore. ***(5 pts)***

4. Write an ARM code to realize a C-subroutine **int strcpy(char *src, char *dst)** which copies a string from the memory location pointed by **src** to another location pointed by **dst**. The return value of this subroutine is the length of the string that has been copied. Your program has to follow the APCS standard. ***(14 pts)***

5. For the following simple assembly code: ***(13 pts)***
   (a) Explain the function of this code. ***(5 pts)***
   (b) Describe the main drawback of this code. Write a more efficient code than can implement the same function. ***(8 pts)***

```
        BL      TAB
TAB     CMP     r0, #0
        BEQ     SUB0
        CMP     r0, #1
        BEQ     SUB1
        ……..
        CMP     r0, #9
        BEQ     SUB9
```

6. The instruction coding of Thumb data processing instructions is shown in the following figure. ***(21 pts)***

(a) Check if the following Thumb instruction syntax is correct. If not, you should also explain why. ***(12 pts)***

(1) ADD r13, r1, #21

(2) MOV r0, r9

(3) SUBEQ r1, r2, r3

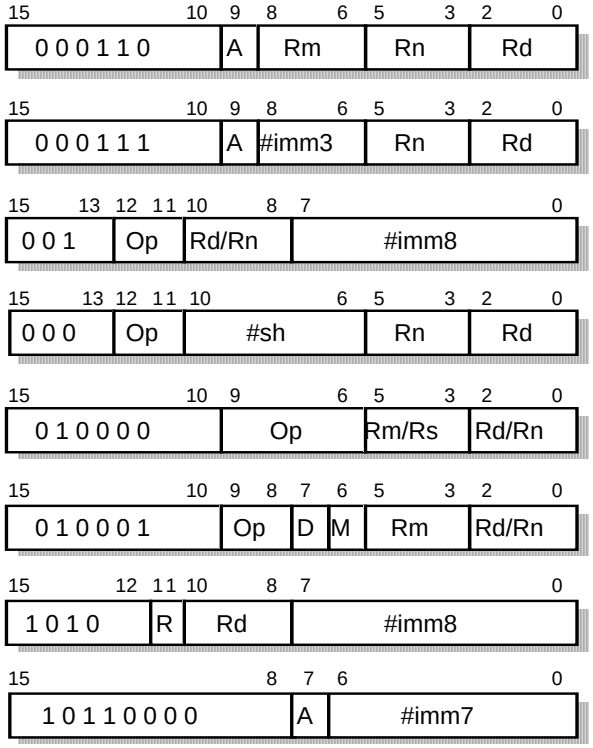
(4) CMP r4, #43

(b) Write the equivalent 32-bit ARM instruction for the following Thumb instruction: ***(9 pts)***

(1) PUSH {r4, r0, lr}

(2) SUB r3, #52

(3) LSR r1, r3, #3



(1) `ADD|SUB Rd,Rn,Rm`

(2) `ADD|SUB Rd,Rn,#imm3`

(3) `<Op> R d/Rn ,#imm8`

(4) `LSL|LSR|ASR Rd,Rn,#shift`

(5) `<Op> Rd/Rn,Rm/Rs`

(6) `ADD|CMP|MOV Rd/Rn,Rm`

(7) `ADD Rd,SP|PC,#imm8`

(8) `ADD|SUB SP,SP,#imm7`

7. For the thumb instruction, ***(10 pts)***

(c) Fill a correct instruction in the empty box in the program shown in the right such that it can call the subroutine written in Thumb instruction correctly. ***(4 pts)***

(d) Translate the following ARM code into the THUMB code using the fewest number of instructions such that it can perform the division correctly. ***(6 pts)***

```
        CODE 32
        [          ]
        BLX     r0

        CODE 16
Thumb   ADD     r1, #1
        BX      lr
```

```
        MOV r3, #0
Loop    SUBS    r0, r0, r1
        ADDGE   r3, r3, #1
        BGE     Loop
        ADD     r2, r0, r1
```

8. Find out the 32-bit instruction coding for the following ARM instructions based on the given coding information. (The coding **P, U, W, L** bits in multiple-register-transfer instructions is the same as single-register transfer instructions.) ***(12 pts)***

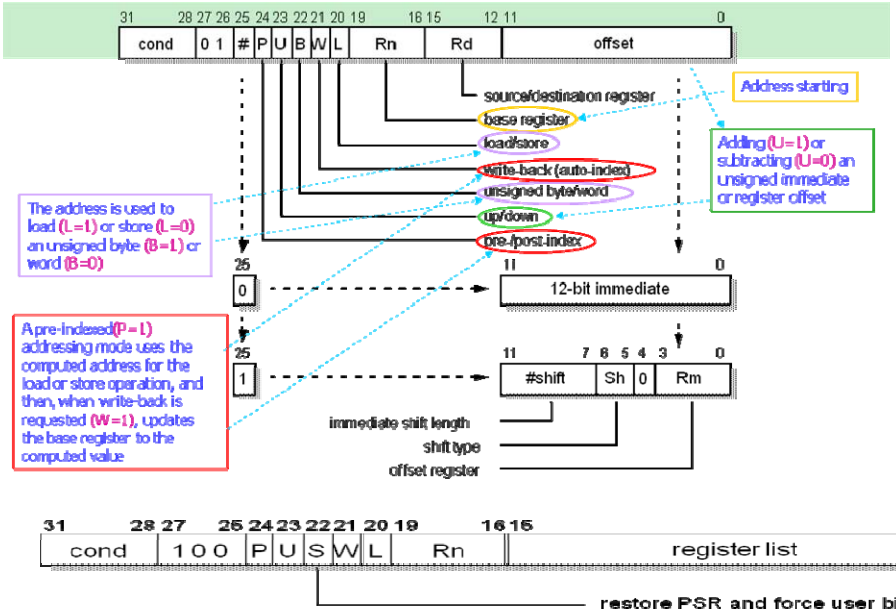
   (a) LDRB r9, [r1, r7, LSR #2]!

   (b) STRGE r1, [r2], #-8

   (c) LDMFD sp!, [r3,r1,r10-r12]

Coding table of Shift Operation

00	LSL	01	LSR	10	ASR	11	ROR



Opcode [31:28]	Interpretation
0000	Equal / equals zero
0001	Not equal
0010	Carry set / unsigned higher or same
0011	Carry clear / unsigned lower
0100	Minus / negative
0101	Plus / positive or zero
0110	Overflow
0111	No overflow
1000	Unsigned higher
1001	Unsigned lower or same
1010	Signed greater than or equal
1011	Signed less than
1100	Signed greater than
1101	Signed less than or equal
1110	Always
1111	Never (do not use!)

9. Figure 7(b) shows the partial assembly code corresponding to the original C code shown in Fig. 7(a). Complete the assembly code by filling the seven space regions. ***(10 pts)***

```
#include <stdio.h>

void func1 (int a);
void func2 (int b);


int main()
{
                int a, b;
                int x[10];

                a = 6;
                b = 7;
                x[8]=30;

                func1(a);
                func2(b);

        printf("x[8]= %d \n",x[8]);

        return 0;
}

void func1 (int a)
{
        int y[5], b;

        b = y[1]+a;
        func2 (b);

}

void func2 (int b)
{
}
```

```
func2
  $a
  .text
      0x00000000:    e1a0f00e    ....
  func1
      0x00000004:    e24dd014    ..M.    SUB     r13,r13,#0x14
      0x00000008:    e59d1004    ....    LDR     r1,[r13,#4]
      0x0000000c:    e0810000    ....    ADD     r0,r1,r0
      0x00000010:    e28dd014    ....    ADD     r13,r13,#0x14
      0x00000014:    eafffffe    ....    B       func2  ; 0x0
  main
      0x00000018:    e52de004    ..-.    STR     r14,[r13,#-4]!
      0x0000001c:    e24dd02c    ,.M.    SUB     r13,r13,#0x2c
      0x00000020:    e3a00006    ....    MOV     r0,#6
      0x00000024:    e3a02007    . ..    MOV     r2,#7
      0x00000028:    e3a0301e    .0..    MOV     r3,#0x1e
      0x0000002c:    e58d3024    $0..    STR     r3,[r13,#0x24]
      0x00000030:    ebfffffe    ....    BL      func1  ; 0x4
      0x00000034:    e1a00002    ....    MOV     r0,r2
      0x00000038:    ebfffffe    ....    BL      func2  ; 0x0
      0x0000003c:    e1a01003    ....    MOV     r1,r3
      0x00000040:    e28f000c    ....    ADD     r0,pc,#0x54
      0x00000044:    ebfffffe    ....    BL      _printf
      0x00000048:    e3a00000    ....    MOV     r0,#0
      0x0000004c:    e28dd02c    ,...    ADD     r13,r13,#0x2c
      0x00000050:    e49df004    ....    LDR     pc,[r13],#4
  $d
      0x00000054:    5d385b78    x[8]    DCD     1563974520
      0x00000058:    6425203d    = %d    DCD     1680154685
      0x0000005c:    00000a20     ...    DCD     2592
```