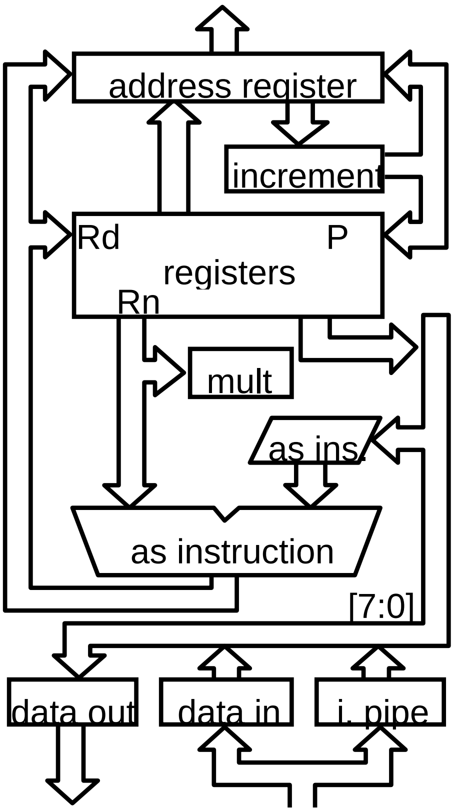
National Sun Yat-Sen University
ASSEMBLY LANGUAGE AND MICROCOMPUTER
Final Exam
2:15-4:15 PM Jan 16 2014

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 ***RSB r0, r1, r2 LSL #2*** at the execution stage. ***(3 pts)***
   (b) Show the datapath activity at each cycle of the execution stage. ***(5 pts)***
   (c) Fill the following immediate field of the instruction used to return from and **IRQ** exception. You have to explain the reason. ***(6 pts)***

SUBS pc, r14, □



2. Suppose an embedded system has an output screen with resolution of 640x480 pixel. We allocate a region of system memory starting from 0xC2000000 as the frame color buffer. Each pixel adopts 32-bit true color format. Write a C or ARM subroutine to clean the screen to white. ***(10 pts)***

3. For the following ARM program, ***(20 pts)***
   (a) Explain the function of this code. ***(4 pts)***
   (b) Translate the following ARM code into the THUMB code by filling the eight blank instructions. ***(10 pts)***
   (c) Explain the effect of **ALIGN** in this ARM code. Also explain why we have to put **ALIGN** here. ***(6 pts)***

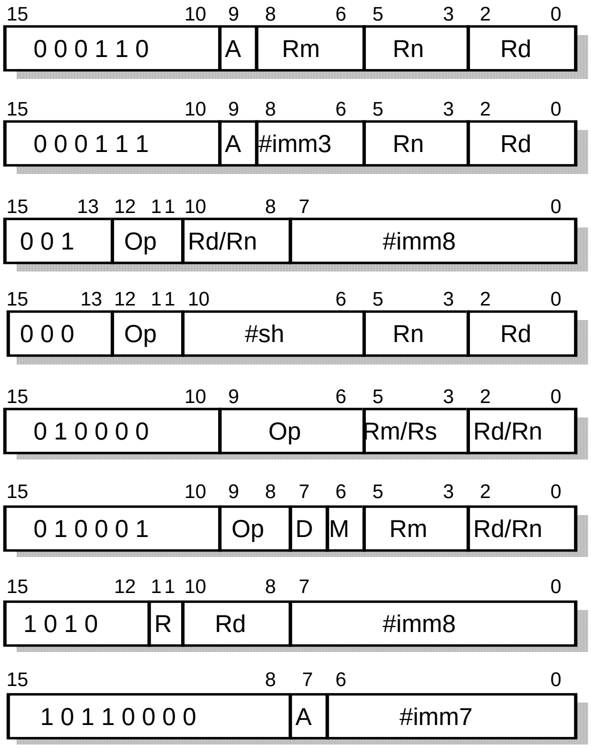
```
            AREA TEST, CODE, READONLY
            ENTRY
START       ADR  r1, TEXT
LOOP        LDRB        r0, [r1], #1
            CMP  r0, #0
            SWINE       #0
            BNE         LOOP
            SWI         #&11
TEXT=            "NSYSU", &0a, &0d,0
            END
```

```
            AREA TEST_THUMB, CODE, READONLY
            ENTRY
            CODE32
            _________   ; get Thumb entry address
            _________   ; enter Thumb area
            CODE16
START       ADR  r1, TEXT
LOOP        _________
            __________
            __________
            __________
            ___________
            ____________
DONE        SWI         #&11
            ALIGN
TEXT=            "NSYSU", &0a, &0d,0
            END
```

4. Write an ARM code to realize a C-subroutine **int strlen(char *src)** which returns the length of the input string. Your program has to follow the APCS standard. ***(10 pts)***

5. Answer the following short questions: ***(8 pts)***
   (a) Briefly describe what **AMBA** is, and its function. ***(4 pts)***
   (b) Describe how CPSR will be affected after executing the following three instructions. ***(4 pts)***
   MRS r0, CPSR   ORR r0, r0, #&20000000   MSR CPSR_f, r0

6. The instruction coding of Thumb data processing instructions is shown in the following figure. ***(18 pts)***
   (a) Check if the following Thumb instruction syntax is correct. If not, you should also explain why. ***(12 pts)***
      (1) SUB r1, r13, #62
      (2) RSB r0, r13
      (3) ADDEQ r1, r2, r3
      (4) PUSH {r3, pc}
   (b) Write the equivalent 32-bit ARM instruction for the following Thumb instruction: ***(6 pts)***
      (1) SUB r3, #24
      (2) LSL r1, r3, #3



15 10 9 8 6 5 3 2 0
0 0 0 1 1 0 | A | Rm | Rn | Rd
15 10 9 8 6 5 3 2 0
0 0 0 1 1 1 | A | #imm3 | Rn | Rd
15 13 12 11 10 8 7 0
0 0 1 | Op | Rd/Rn | #imm8
15 13 12 11 10 6 5 3 2 0
0 0 0 | Op | #sh | Rn | Rd
15 10 9 6 5 3 2 0
0 1 0 0 0 0 | Op | Rm/Rs | Rd/Rn
15 10 9 8 7 6 5 3 2 0
0 1 0 0 0 1 | Op | D | M | Rm | Rd/Rn
15 12 11 10 8 7 0
1 0 1 0 | R | Rd | #imm8
15 8 7 6 0
1 0 1 1 0 0 0 0 | A | #imm7

(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. 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)***

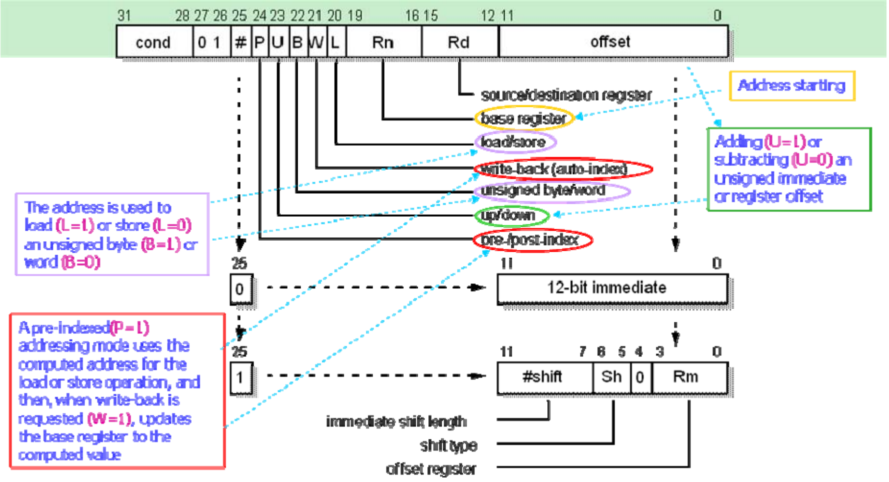
(a) LDRLE r9, [r1, r7, LSR #2]

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

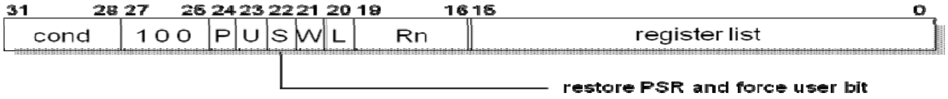
(c) STMED sp!, [r3,r1,r10-r12]

Coding table of Shift Operation

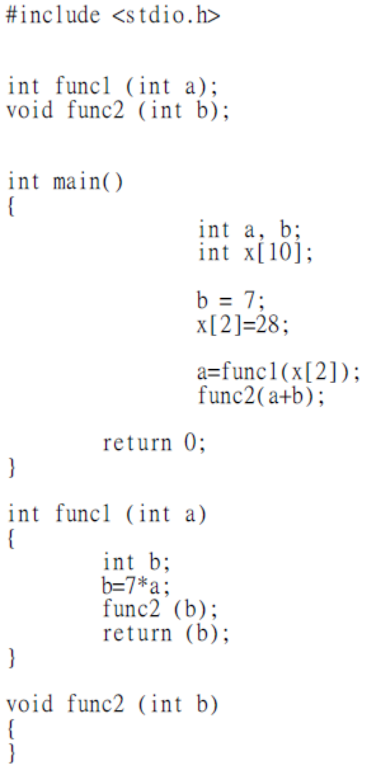
00	LSL	01	LSR	10	ASR	11	ROR


31
28 27 26 25 24 23 22 21 20 19
16 15
12 11
0
cond
0 1
#
P
U
B
W
L
Rn
Rd
offset
source/destination register
base register
load/store
write-back (auto-index)
unsigned byte/word
up/down
pre-/post-index
Address starting
Adding (U=1) or subtracting (U=0) an unsigned immediate or register offset
The address is used to load (L=1) or store (L=0) an unsigned byte (B=1) or word (B=0)
A pre-indexed (P=1) addressing mode uses the computed address for the load or store operation, and then, when write-back is requested (W=1), updates the base register to the computed value
25
0
11
0
12-bit immediate
25
1
11
7 6 5 4 3
0
#shift
Sh
0
Rm
immediate shift length
shift type
offset register

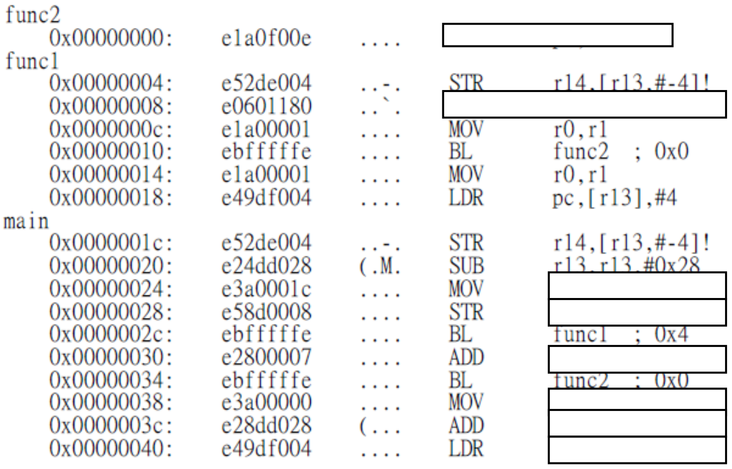
Opcode [31:28]	Interpretation
0000	Equal / equals zero
0001	Not equal
0010	Carry set / unsigned higher or same
0011	Carry clear / unsigned lower
0100	Minus / negative
0101	Plus / positive or zero
0110	Overflow
0111	No overflow
1000	Unsigned higher
1001	Unsigned lower or same
1010	Signed greater than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



8. Complete the eight space regions of the following assembly code which is the disassembled result of the C code shows as below. ***(16 pts)***



```
#include <stdio.h>

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

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

                b = 7;
                x[2]=28;

                a=func1(x[2]);
                func2(a+b);

        return 0;
}

int func1 (int a)
{
        int b;
        b=7*a;
        func2 (b);
        return (b);
}

void func2 (int b)
{
}
```



```
func2
    0x00000000:    e1a0f00e    ....    
func1
    0x00000004:    e52de004    ..-.    STR     r14,[r13,#-4]!
    0x00000008:    e0601180    ..`.    
    0x0000000c:    e1a00001    ....    MOV     r0,r1
    0x00000010:    ebfffffe    ....    BL      func2  ; 0x0
    0x00000014:    e1a00001    ....    MOV     r0,r1
    0x00000018:    e49df004    ....    LDR     pc,[r13],#4
main
    0x0000001c:    e52de004    ..-.    STR     r14,[r13,#-4]!
    0x00000020:    e24dd028    (.M.    SUB     r13,r13,#0x28
    0x00000024:    e3a0001c    ....    MOV     
    0x00000028:    e58d0008    ....    STR     
    0x0000002c:    ebfffffe    ....    BL      func1  ; 0x4
    0x00000030:    e2800007    ....    ADD     
    0x00000034:    ebfffffe    ....    BL      func2  ; 0x0
    0x00000038:    e3a00000    ....    MOV     
    0x0000003c:    e28dd028    (...    ADD     
    0x00000040:    e49df004    ....    LDR     
```