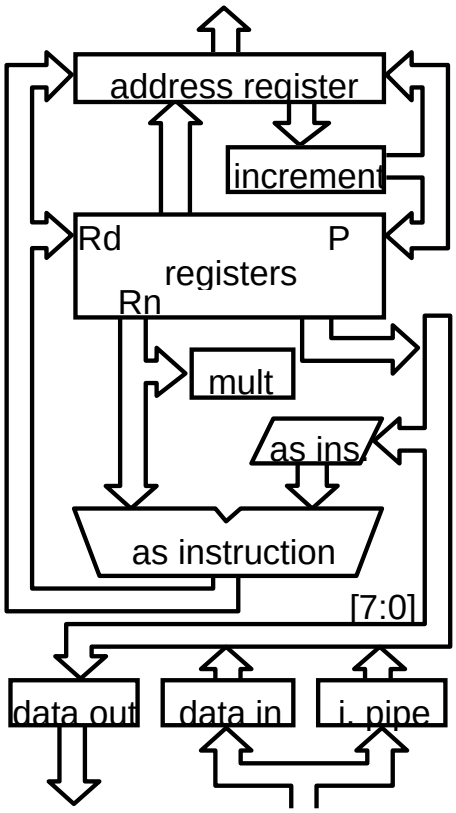
National Sun Yat-Sen University
ASSEMBLY LANGUAGE AND MICROCOMPUTER
Final Exam
9:20-11:30AM

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. ***(10 pts)***
   (a) Find out the number of cycles it will takes to run the ARM instruction ***STR r0, [r1], #4*** at the execution stages. ***(4 pts)***
   (b) Show the datapath activity at each cycle. ***(6 pts)***



2. Show how the variable ***data*** is organized in the little-endian memory without packing. ***(10 pts)***

```
struct T1 {float a[2], short b};
struct T2 { T1 c , char d2[5] } data;
```

3. Answer the following short questions: ***(9 pts)***
   (a) Explain what "**callee-saved**" register variables mean. ***(3 pts)***
   (b) Explain what the "**stack**" is and list three different usages of stack to support the high-level programming. ***(6 pts)***

4. For the following simple assembly code: ***(15 pts)***
   (a) Explain the function of this code. ***(5 pts)***
   (b) Describe the main drawback of this realization. Rewrite and improve this code to solve the drawback. ***(10 pts)***

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

5. For the 32-bit ARM instruction set: ***(20 pts)***

(a) Write a C subroutine that matches the code shown in the right. ***(8 ts)***

(b) Write an ARM program to implement a positive integer division subroutine ***div(int A, int D, int* R)*** which returns the quotient of ***A*** divided by ***D***. The remainder of this division has to be written to the memory location pointed by ***R***. You should try to use as fewer instructions as possible. ***(12 ts)***

```
SubRt
        MOV     r2, r0;
        MOV     r0, #0;
        MOV     r1, #0x10;
LOOP    LDR     r3, [r2], #4
        SUBS    r1, r1, #1
        ADD     r0, r3, r0
        BNE     LOOP
        BX      r14
```

6. The instruction coding of Thumb data processing instructions is shown in the following figure.

(a) Check if the following Thumb instruction syntax is correct. If not, you should also explain why. ***(10 pts)***

(1) ADD r3, r8, r2

(2) SUBNE r2, #7

(3) CMP r4, r9

(4) POP r3, lr

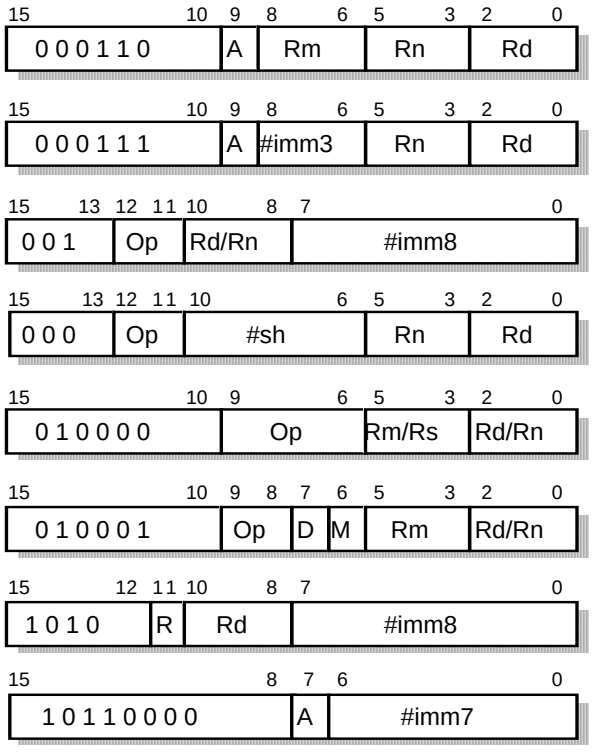
(5) BLGT Subroutine (* Subroutine represents a symbol/label in the program)

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

(1) PUSH r4, r0, lr

(2) SUB r3, #62

(3) LSR r2, r3



7. Find out the 32-bit instruction coding for the following ARM instructions based on the given coding information. ***(12 pts)***

   (a) STRB r7, [r1, r0, LSL #2]

   (b) LDREQ r4, [r2, #-8]!

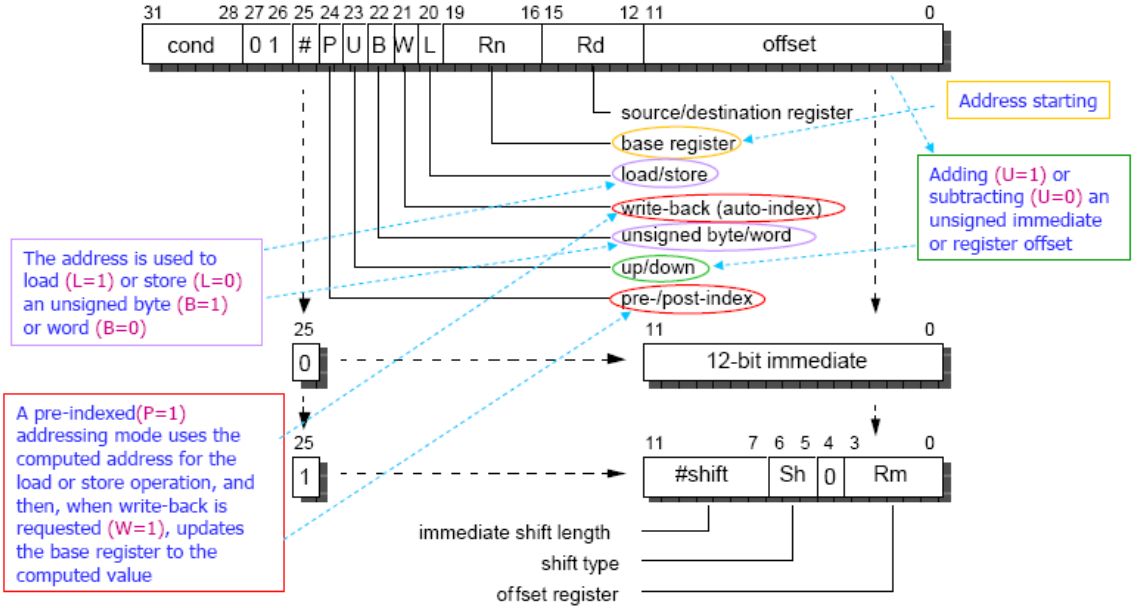
   (c) LDR r1, [r9], #4.

Coding table of ARM condition codes

0000	EQ	0100	MI	1000	HI	1100	GT
0001	NE	0101	PL	1001	LS	1101	LE
0010	CS/HS	0110	VS	1010	GE	1110	AL
0011	CC/LO	0111	VC	1011	LT	1111	NV

Coding table of Shift Operation

00	LSL	01	LSR	10	ASR	11	ROR



LDRLS pc, [r1, r0, LSL #2] Chap.3, p.35

8. 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 ten space regions. ***(15 pts)***

```
#include <stdio.h>

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

int main()
{       int a, b, c, d, e;
                char data[20];

                a = 6;
                b = 7;
                c = 10;

                d=func1(c);
                e=func2(a, b);

        printf("Result : %s %d %d\n",data, d, e);

        return 0;
}

int func1 (int a)
{
        int c[10], b;

        b = c[2]+a;

        return(b);
}
```

```
func1
$a
.text
    0x00000000:    SUB      r13,r13,#0x28
    0x00000004:    LDR      _____________
    0x00000008:    ADD      r0,r1,r0
    0x0000000c:    ADD      _____________
    0x00000010:    MOV      _____________
main
    0x00000014:    STMFD    _____________
    0x00000018:    SUB      r13,r13,#0x18
    0x0000001c:    MOV      r2,#6
    0x00000020:    MOV      r3,#7
    0x00000024:    MOV      r0,#0xa
    0x00000028:    BL       func1   ; 0x0
    0x0000002c:    MOV      r4,r0
    0x00000030:    MOV      _____________
    0x00000034:    MOV      _____________
    0x00000038:    BL       func2
    0x0000003c:    MOV      _____________
    0x00000040:    MOV      _____________
    0x00000044:    ADD      r1,r13,#4
    0x00000048:    ADD      r0,pc,#0xc ; #0x5c
    0x0000004c:    BL       _printf
    0x00000050:    MOV      _____________
    0x00000054:    ADD      r13,r13,#0x18
    0x00000058:    ____________________________
$d
    0x0000005c:    DCD      1970496850
    0x00000060:    DCD      975205484
    0x00000064:    DCD      544417056
    0x00000068:    DCD      622879781
    0x0000006c:    DCD      2660
```