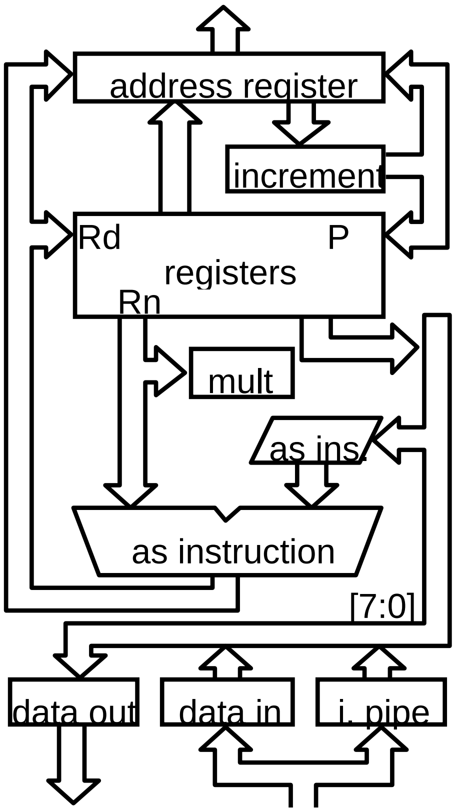
National Sun Yat-Sen University
ASSEMBLY LANGUAGE AND MICROCOMPUTER
Final Exam
1:15-3:15 PM Jan 12 2012

Name: ___________________

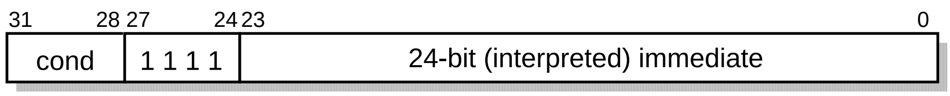
Note: Although there are more than 100 points for this exam, the maximum score you can get is 100 points.

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 ***SUB r0, r1, r2 LSL #12*** at the execution stages. ***(3 pts)***
   (b) Show the datapath activity at each cycle. ***(5 pts)***
   (c) Complete the following ARM instruction to return from the data abort exception. You have to explain the reason. ***(6 pts)***

   SUBS pc, ________



2. The instruction coding of ***SWI*** instruction is shown as follows.
   Write the software interrupt handler subroutine assembly code that, after invoked, will return a value in ***r0*** which equals the software interrupt number plus 1. For example, after executing **swi #8**, ***r0*** becomes 9. You don't have to provide the code about the installation of this handler. You just have to provide the handler subroutine itself. ***(12 pts)***

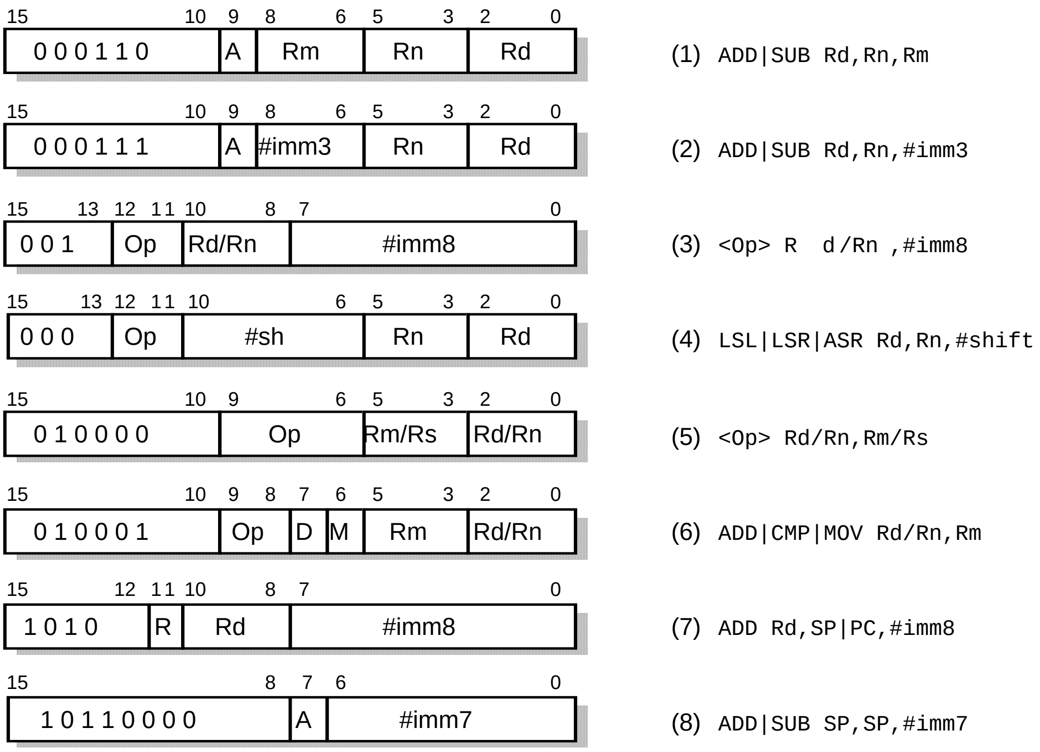


3. Suppose one ARM-based embedded system uses total of 1K-bytes ROM and 256K-bytes of SRAM memory. ***(13 pts)***
   (a) Discuss which memory module (ROM or RAM) the low addresses are usually mapped to. ***(3 pts)***
   (b) Draw the block diagram of the ARM memory system. The detailed I/O signals of the memory modules and associated read-write control signals should be clearly drawn. However, you don't have to draw the detailed logic circuit used to generate those control signals. ***(10 pts)***

4. Write an **<u>Thumb</u>** code to realize a C-subroutine **<u>int strcpy(char *src, char *dst)</u>** 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. ***<u>(14 pts)</u>***

5. Write an short sequence of a ARM code based on ***<u>BX</u>*** instruction to call the **<u>strcpy</u>** subroutine which is implemented by **<u>Thumb</u>** code. (Note that the **<u>strcpy</u>** subroutine will return to the caller function after execution.) ***<u>(4 pts)</u>***

6. The instruction coding of Thumb data processing instructions is shown in the following figure. ***<u>(21 pts)</u>***
   (a) Check if the following Thumb instruction syntax is correct. If not, you should also explain why. ***<u>(12 pts)</u>***
      (1) SUB r8, r1, #21
      (2) CMP r0, r9
      (3) ADD r1, r2, r3, LSR #2
      (4) SUBEQ SP,SP, #43
   (b) Write the equivalent 32-bit ARM instruction for the following Thumb instruction: ***<u>(9 pts)</u>***
      (1) POP {r4, r0}
      (2) SUB r3, #52
      (3) ASR r1, r3, #3



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)***

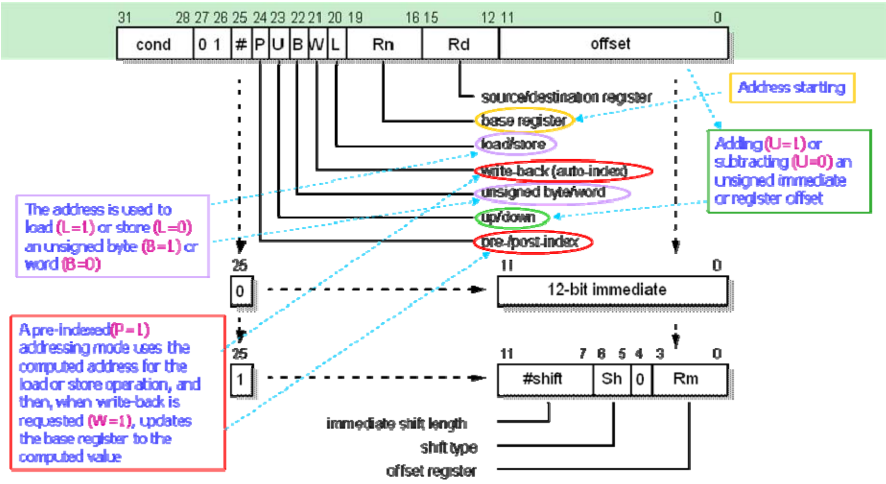
(a) STRNE r9, [r1, r7, LSR r2]

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

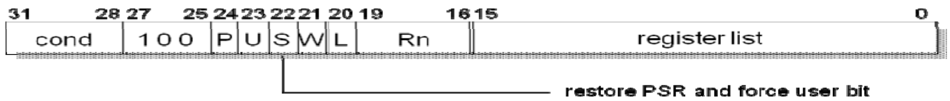
(c) STMEA sp!, [r3,r1,r10-r12]

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!)



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    [          ]
```