COSE321 Computer Systems Design
Final Exam, Spring 2019

Name: _Solutions_

Note: No Explanations, No Credits!

1. Answer to the following questions about Thumb2 **(35 points)**
   a. Convert the following C code to **Thumb** assembly code **using the `IT` instruction**. Show how the `CPSR'IT` field changes as your answer code gets executed. Refer to the ARM condition table **(20 points = 10 + 10)**

```
int  f, g, h, i, j;

if (i <= j) {
    f = g + h;
    f++;
} else {
    f = g – h
    f--;
}
```

```
// Assembly code here
// Assume that R0 = f,  R1 = g, R2 = h, R3 = i, R4 = j

        cmp  R3, R4;
        ITTEE   LE;
        addle  R0, R1, R2
        addle  R0, R0, #1
        subgt  R0, R1, R2;
        subgt  R0, R0, #1;
```

```
// Show CPSR'IT as your answer code gets executed.

        cmp  R3, R4;
       ITTEE  LE;              //  IT = 8'b1101_1001 (0xD9)
       addle  R0, R1, R2       //  IT = 8'b1101_0010 (0xD2)
       addle  R0, R0, #1       //  IT = 8'b1100_0100 (0xC4)
       subgt  R0, R1, R2;      //  IT = 8'b1100_1000 (0xC8)
       subgt  R0, R0, #1;      //  IT = 8'b0000_0000 (0x00)
```

Opcode [31:28]	Mnemonic extension	Meaning	Condition flag state
0000	EQ	Equal	Z set
0001	NE	Not equal	Z clear
0010	CS/HS	Carry set/unsigned higher or same	C set
0011	CC/LO	Carry clear/unsigned lower	C clear
0100	MI	Minus/negative	N set
0101	PL	Plus/positive or zero	N clear
0110	VS	Overflow	V set
0111	VC	No overflow	V clear
1000	HI	Unsigned higher	C set and Z clear
1001	LS	Unsigned lower or same	C clear or Z set
1010	GE	Signed greater than or equal	N set and V set, or N clear and V clear (N == V)
1011	LT	Signed less than	N set and V clear, or N clear and V set (N != V)
1100	GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear (Z == 0,N == V)
1101	LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set (Z == 1 or N != V)

b. In the following Thumb2 code, **which instructions in the IT block** are executed and **why**? **What values** do **R0, R1, and R2** have after execution? **Explain** with CPSR'IT and N,Z,C,V in CPSR. **(15 points)**

// At the beginning, R0 = 0, R1 = 1, R2 = 2, R3 =3, R4 = 4

```
        cmp  R4, R3;
        ITTEE  GT;
        cmpgt  R3, R3
        addgt  R0, R0, #1
        addle  R1, R1, #2;
        addle  R2, R2, #3;
```

```
        cmp  R4, R3;          //  NZCV = 0010 (<= 4 – 3)
        ITTEE  GT;            //  IT = 8'b1100_0111 (0xC7)
        cmpgt  R3, R3         //  IT = 8'b1100_1110 (0xCE)
                              //  NZCV = 0110 (<= 3 – 3)
        addgt  R0, R0, #1     //  IT = 8'b1101_1100 (0xDC)
        addle  R1, R1, #2;    //  IT = 8'b1101_1000 (0Xd8)
        addle  R2, R2, #3;    //  IT = 8'b0000_0000 (0x00)
```

Instructions in blue are executed
After execution, R0 = 0, R1 = 3, R2 = 5

2. Answer to the following questions about interrupt in ARM (**40 points**)

a. Explain the difference between the following instructions. When would you use the instructions? (**15 points = 5 + 5 + 5**)

	Detailed operations	Usage case
`subs r1, r2, #0;`	• r1 ← r2 – 0 • N,Z,C, V updated	normal arithmetic operation for conditional execution
`sub pc, lr, #0;`	• pc ← lr	Return from a function
`subs pc, lr, #4;`	• pc ← lr - 4 • cpsr ← spsr	Return from exception or interrupt

b. Inter-Processor Interrupt (IPI) is a special type of interrupt by which one processor may interrupt another processor in a multiprocessor system. There are 2 CPUs in Zynq. You want to design a system where CPU0 informs some event (through interrupt #7) to CPU1 via IPI. Write an ARM assembly code by referring to the tables in the following pages (**10 points**)

```
ldr   r0, = GICD_SGIR
mov  r1, #0 << 24   //  TargetListFilter = 0b00
mov  r2, #0x02 << 16   // CPUTargetList = 0b0000_00010
mov  r3, #7   // SGINTID = 7
orr   r1, r1, r2
orr   r1, r1, r3
str   r1, [r0]
```

Table 4-1 Distributor register map

Offset	Name	Type	Reset[a]	Description
0x000	GICD_CTLR	RW	0x00000000	Distributor Control Register
0x004	GICD_TYPER	RO	IMPLEMENTATION DEFINED	Interrupt Controller Type Register
0x008	GICD_IIDR	RO	IMPLEMENTATION DEFINED	Distributor Implementer Identification Register
0x00C-0x01C	-	-	-	Reserved
0x020-0x03C	-	-	-	IMPLEMENTATION DEFINED registers
0x040-0x07C	-	-	-	Reserved
0x080	GICD_IGROUPRn[b]	RW	IMPLEMENTATION DEFINED[c]	Interrupt Group Registers
0x084-0x0FC			0x00000000	
0x100-0x17C	GICD_ISENABLERn	RW	IMPLEMENTATION DEFINED	Interrupt Set-Enable Registers
0x180-0x1FC	GICD_ICENABLERn	RW	IMPLEMENTATION DEFINED	Interrupt Clear-Enable Registers
0x200-0x27C	GICD_ISPENDRn	RW	0x00000000	Interrupt Set-Pending Registers
0x280-0x2FC	GICD_ICPENDRn	RW	0x00000000	Interrupt Clear-Pending Registers
0x300-0x37C	GICD_ISACTIVERn[d]	RW	0x00000000	GICv2 Interrupt Set-Active Registers
0x380-0x3FC	GICD_ICACTIVERn[e]	RW	0x00000000	Interrupt Clear-Active Registers
0x400-0x7F8	GICD_IPRIORITYRn	RW	0x00000000	Interrupt Priority Registers
0x7FC	-	-	-	Reserved
0x800-0x81C	GICD_ITARGETSRn	RO[f]	IMPLEMENTATION DEFINED	Interrupt Processor Targets Registers
0x820-0xBF8		RW[f]	0x00000000	
0xBFC	-	-	-	Reserved
0xC00-0xCFC	GICD_ICFGRn	RW	IMPLEMENTATION DEFINED	Interrupt Configuration Registers
0xD00-0xDFC	-	-	-	IMPLEMENTATION DEFINED registers
0xE00-0xEFC	GICD_NSACRn[e]	RW	0x00000000	Non-secure Access Control Registers, optional
0xF00	GICD_SGIR	WO	-	Software Generated Interrupt Register
0xF04-0xF0C	-	-	-	Reserved
0xF10-0xF1C	GICD_CPENDSGIRn[e]	RW	0x00000000	SGI Clear-Pending Registers
0xF20-0xF2C	GICD_SPENDSGIRn[e]	RW	0x00000000	SGI Set-Pending Registers
0xF30-0xFCC	-	-	-	Reserved
0xFD0-0xFFC	-	RO	IMPLEMENTATION DEFINED	*Identification registers* on page 4-119

a. For details of any restrictions that apply to the reset values of IMPLEMENTATION DEFINED cases see the appropriate register description.
b. In a GICv1 implementation, present only if the GIC implements the GIC Security Extensions, otherwise RAZ/WI.
c. For more information see *GICD_IGROUPR0 reset value* on page 4-92.
d. In GICv1, these are the Active Bit Registers, ICDABRn. These registers are RO.

Figure 4-17 shows the GICD_SGIR bit assignments.

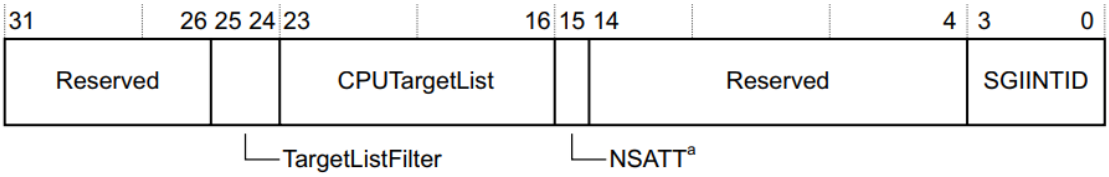


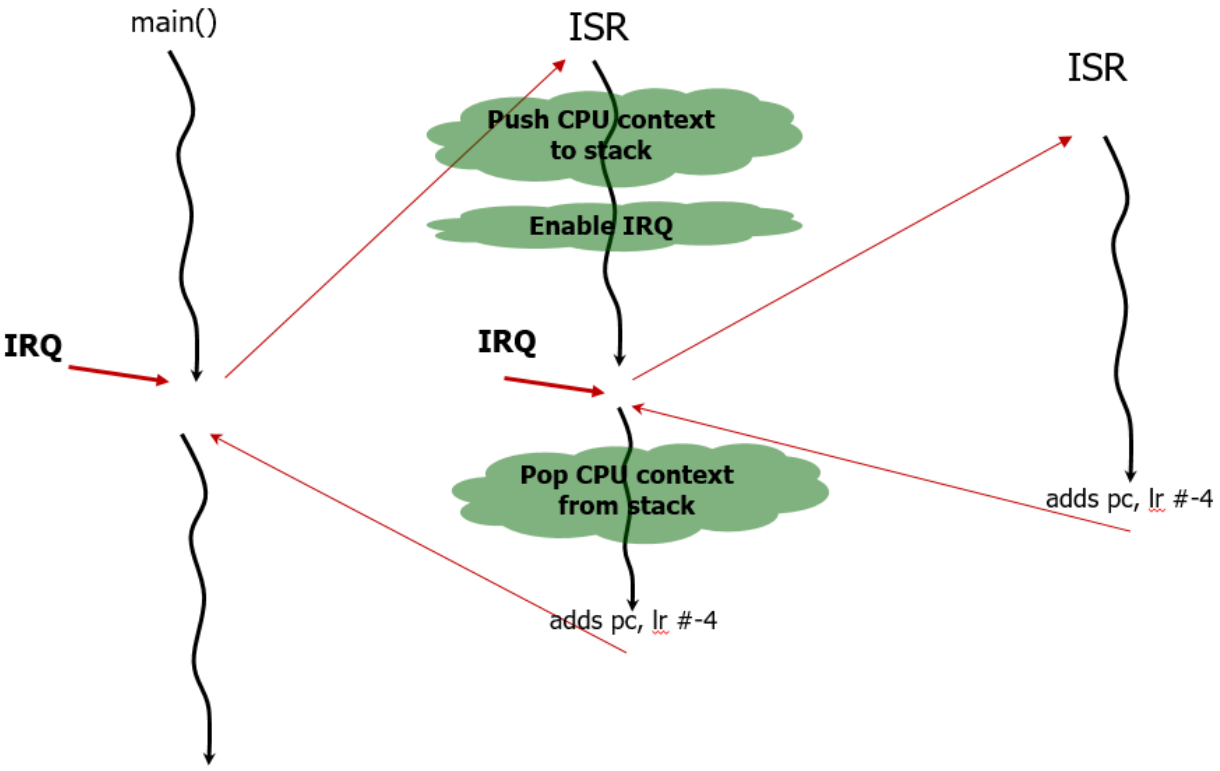
Figure 4-17 GICD_SGIR bit assignments

Table 4-21 GICD_SGIR bit assignments

Bits	Name	Function
[31:26]	-	Reserved.
[25:24]	TargetListFilter	Determines how the distributor must process the requested SGI: `0b00` Forward the interrupt to the CPU interfaces specified in the CPUTargetList field[a]. `0b01` Forward the interrupt to all CPU interfaces except that of the processor that requested the interrupt. `0b10` Forward the interrupt only to the CPU interface of the processor that requested the interrupt. `0b11` Reserved.
[23:16]	CPUTargetList	When TargetList Filter = `0b00`, defines the CPU interfaces to which the Distributor must forward the interrupt. Each bit of CPUTargetList[7:0] refers to the corresponding CPU interface, for example CPUTargetList[0] corresponds to CPU interface 0. Setting a bit to 1 indicates that the interrupt must be forwarded to the corresponding interface. If this field is `0x00` when TargetListFilter is `0b00`, the Distributor does not forward the interrupt to any CPU interface.
Bits	**Name**	**Function**
[15]	NSATT	Implemented only if the GIC includes the Security Extensions. Specifies the required security value of the SGI: **0** Forward the SGI specified in the SGIINTID field to a specified CPU interface only if the SGI is configured as Group 0 on that interface. **1** Forward the SGI specified in the SGIINTID field to a specified CPU interfaces only if the SGI is configured as Group 1 on that interface. This field is writable only by a Secure access. Any Non-secure write to the GICD_SGIR generates an SGI only if the specified SGI is programmed as Group 1, regardless of the value of bit[15] of the write. See *SGI generation when the GIC implements the Security Extensions* for more information. **Note** If GIC does not implement the Security Extensions, this field is reserved.
[14:4]	-	Reserved, SBZ.
[3:0]	SGIINTID	The Interrupt ID of the SGI to forward to the specified CPU interfaces. The value of this field is the Interrupt ID, in the range 0-15, for example a value of `0b0011` specifies Interrupt ID 3.

a. When TargetListFilter is `0b00`, if the CPUTargetList field is `0x00` the Distributor does not forward the interrupt to any CPU interface.

c. The figure below shows the rough structure of the code for the **nested** interrupt (IRQ) processing. **Write** an ARM assembly code for the **push**, **pop** and **interrupt-enable** operations. **Why** do you have to push and pop the CPU context? (**15 points = 12 + 3**)



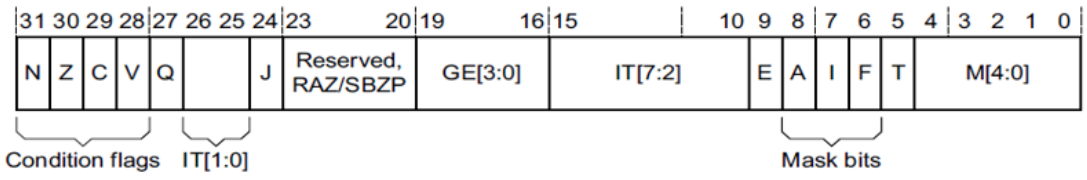
Because the nested interrupt corrupts the registers (r0~r12, lr, SPSR)

Push operation	Pop operation	Enable IRQ
`// Assume full-descending stack`		

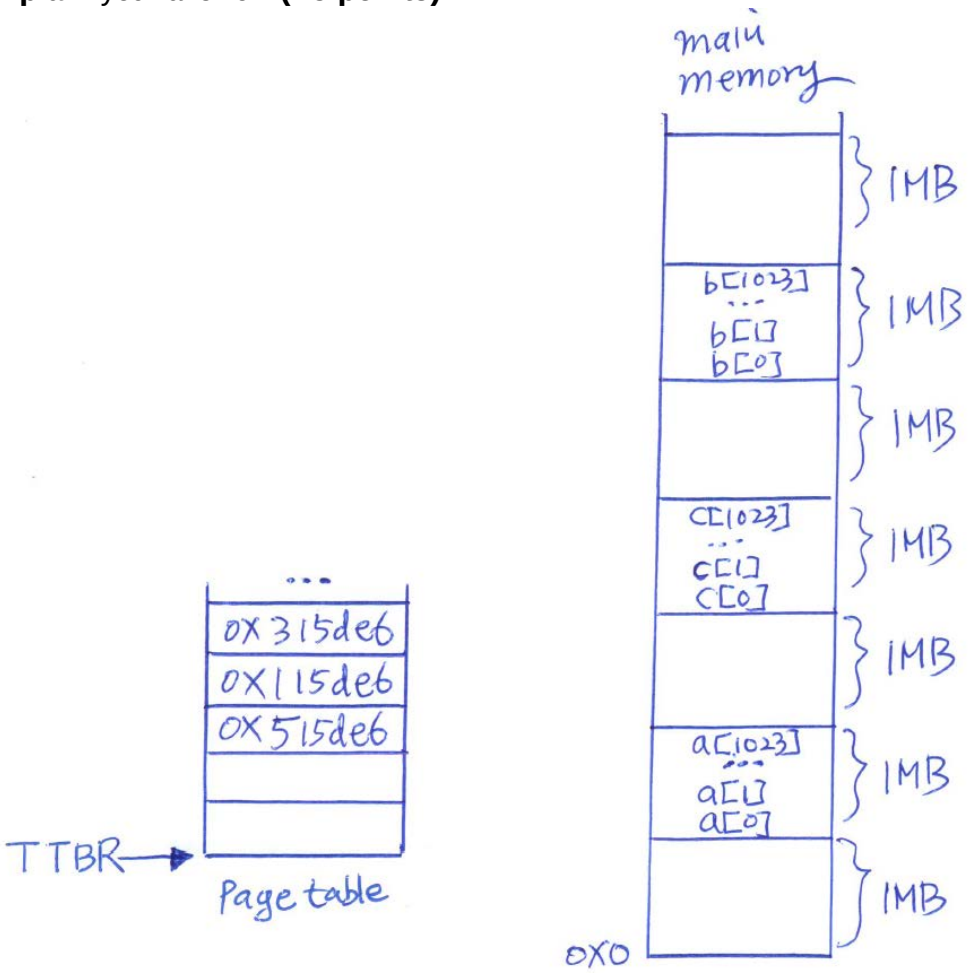
`// push r0 ~ r12`
`stmfd sp!, {r0 ~ r12}`
 	
`// push lr, SPSR`
`srsfd sp!, #0x12`
	`// pop r0 ~ r12`
	`ldmfd sp!, {r0 ~ r12}`
	
	`// pop lr, SPSR`
	`rfefd sp!, #0x12`
		`// Turn off I bit in CPSR`
		
		`cpsIE i`

Format of the CPSR and SPSRs

The CPSR and SPSR bit assignments are:



3. There are 3 arrays of 1024 integers stored in memory, as shown below: `a[1024]`, `b[1024]`, and `c[1024]`. The MMU is enabled for the virtual memory, and the page table is shown below as well. **Write** an ARM assembly code that does `c[i] = a[i] + b[i] for i = 0 to 1023`. **Explain** your answer. **(25 points)**




```
        mov r0, #0x40_0000   // &c[0]
        mov r1, #0x30_0000   // &a[0]
        mov r2, #0x20_0000   // &b[0]

        mov r3, #1024
loop:
        ldr r4, [r1], #4    // post-indexed addressing
        ldr r5, [r2], #4
        add r6, r4, r5
        str r6, [r0], #4
        subs r3, r3, #1
        bne  loop
```

Short-descriptor translation table first-level descriptor formats

Each entry in the first-level table describes the mapping of the associated 1MB MVA range.

Figure B3-4 shows the possible first-level descriptor formats.

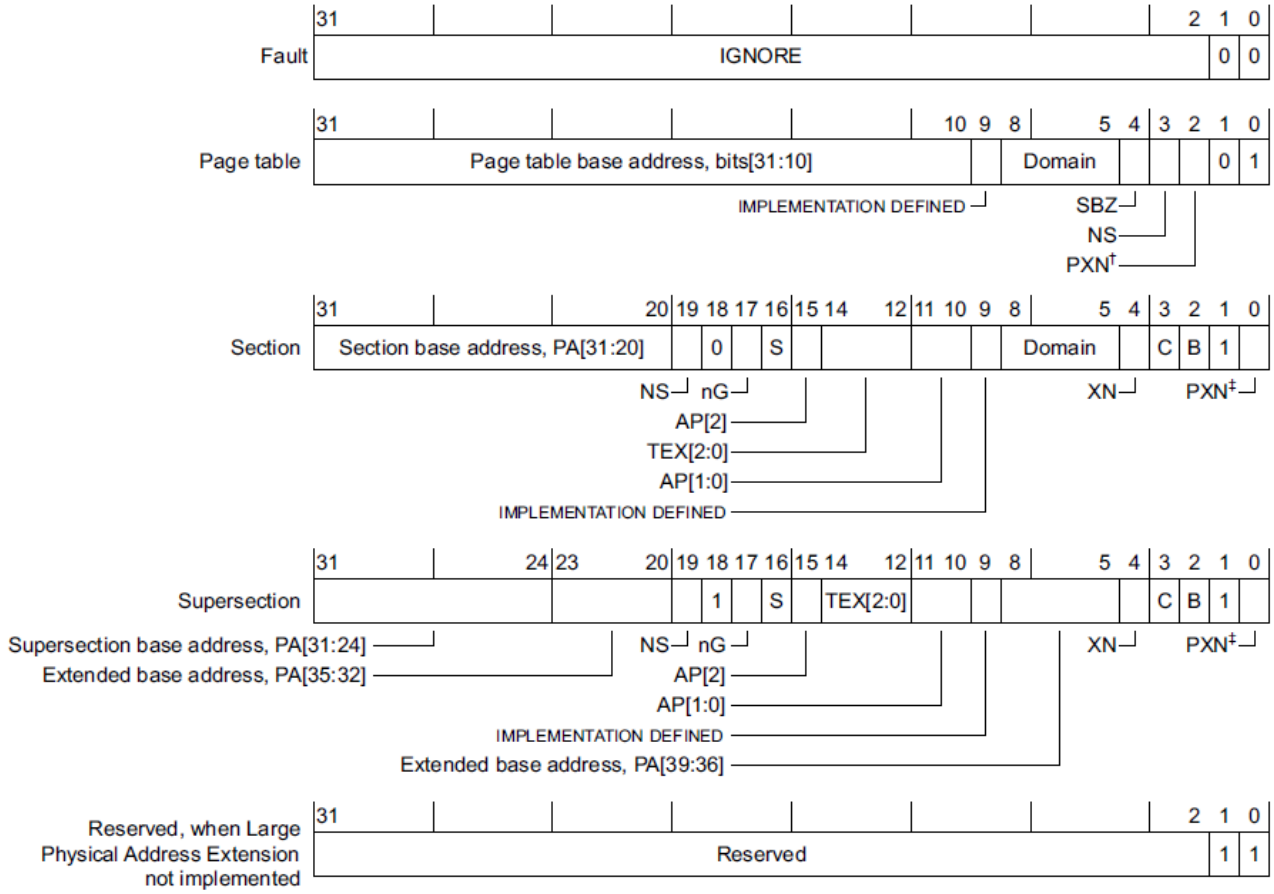


Figure B3-4 Short-descriptor first-level descriptor formats