COSE321 Computer Systems Design
Final Exam, Spring 2020

Name: _____

Note: No Explanations, No Credits!

1. Convert the C code to ARM assembly (or vice versa) with the conditional execution. It means that **NO branch instructions** are allowed in assembly. Refer to the ARM condition table in the next page. **(25 points)**

(5 points)

```
int  a, b, c;

if (a == b)  c++;
else         c--;
```

// assume that R0 = a, R1 = b, R2 = c

(10 points)

// assume that R0 = a, R1 = b, R2 = c,
R3 = d, R4 = e

```
cmp    R0, R1
cmpeq  R2, R3
addgt  R4, R4, #1;
sublt  R4, R4, #1;
```

(10 points)

// assume that R0 = a, R1 = b, R2 = c,
R3 = d, R4 = e

```
cmp    R0, R1
subeqs R2, R3, R4
addlo  R2, R2, #1;
subhi  R2, R2, #1;
```

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)

2. Convert the following C code to the ARM assembly. You should follow the ARM calling convention, shown in the next page. According to the calling convection, the arguments in function are passed through r0, r1, r2, and r3. The return values from function are passed through r0 and r1. (**15 points**)

```
int main()
{

   return (foo(4, 3));

}



int foo(int a, int b) {

   int c, d;

   c = a + b;
   d = a - b;

   return (foo2(c, d));

}



int foo2(int a, int b) {

   return (a - b);

}
```

→

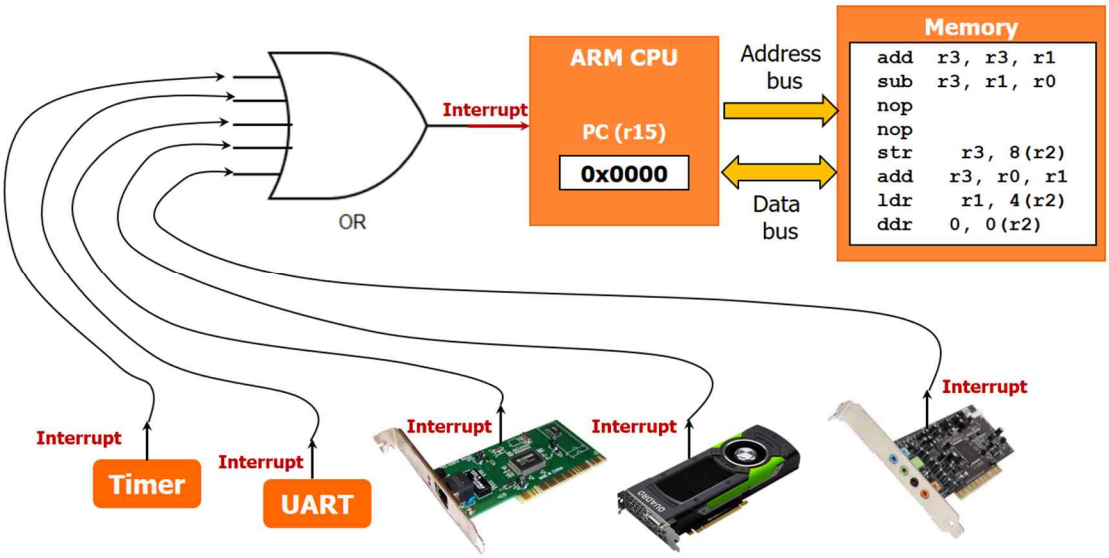
```
main:




foo:




foo2:
```

Table 6.1: Table 2, Core registers and AAPCS usage

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable-register 4.
r6	v3		Variable-register 3.
r5	v2		Variable-register 2.
r4	v1		Variable-register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

3. Answer to the following questions about interrupt (**30 points**)

   a. Assume that the interrupts from I/O devices are connected to CPU through a simple OR gate, as shown below. 1) How would you figure out who requested interrupts in ISR (Interrupt Service Routine)? (**5 points**) 2) How would you design the priority mechanism in ISR? (**5 points**)



b. Now, we have the GIC. Assume that there are 3 I/O devices (USB, UART, and Private Timer) generating normal interrupts (via IRQ connection to Arm CPU). You want to write the Arm assembly program for servicing the Private Timer interrupt. Write an ARM assembly program that jumps to the appropriate handler and services the Private Timer interrupt. In the handler for the Private Timer, toggle the LED connected to GPIO. The Interrupt ID# of the Private Timer is 29. (**20 points**)

```
// Assume that stack pointers are
//                 already set up




#define csd_LED_ADDR   0x41200000




// Interrupt Vector Table
csd_vector_table:
      b .
      b .
      b .
      b .
      b .
      b .
      b csd_IRQ_ISR
      b .









.data
.align 4

led_value: .word 0x5
```

```
// --------
// IRQ ISR
// ---------

csd_IRQ_ISR:









Private_Timer_ISR:
```

<GIC CPU Interface>

Base address: 0xF8F0_0100

Table 4-2 CPU interface register map

Offset	Name	Type	Reset	Description
0x0000	GICC_CTLR	RW	0x00000000	CPU Interface Control Register
0x0004	GICC_PMR	RW	0x00000000	Interrupt Priority Mask Register
0x0008	GICC_BPR	RW	0x0000000x[a]	Binary Point Register
0x000C	GICC_IAR	RO	0x000003FF	Interrupt Acknowledge Register
0x0010	GICC_EOIR	WO	-	End of Interrupt Register
0x0014	GICC_RPR	RO	0x000000FF	Running Priority Register
0x0018	GICC_HPPIR	RO	0x000003FF	Highest Priority Pending Interrupt Register
0x001C	GICC_ABPR[b]	RW	0x0000000x[a]	Aliased Binary Point Register
0x0020	GICC_AIAR[c]	RO	0x000003FF	Aliased Interrupt Acknowledge Register
0x0024	GICC_AEOIR[c]	WO	-	Aliased End of Interrupt Register
0x0028	GICC_AHPPIR[c]	RO	0x000003FF	Aliased Highest Priority Pending Interrupt Register
0x002C-0x003C	-	-	-	Reserved
0x0040-0x00CF	-	-	-	IMPLEMENTATION DEFINED registers
0x00D0-0x00DC	GICC_APRn[c]	RW	0x00000000	Active Priorities Registers
0x00E0-0x00EC	GICC_NSAPRn[c]	RW	0x00000000	Non-secure Active Priorities Registers
0x00ED-0x00F8	-	-	-	Reserved
0x00FC	GICC_IIDR	RO	IMPLEMENTATION DEFINED	CPU Interface Identification Register
0x1000	GICC_DIR[c]	WO	-	Deactivate Interrupt Register

a. See the register description for more information.
b. Present in GICv1 if the GIC implements the GIC Security Extensions. Always present in GICv2.
c. GICv2 only.

Figure 4-27 shows the IAR bit assignments.

Figure 4-27 GICC_IAR bit assignments

Table 4-34 shows the IAR bit assignments.

Table 4-34 GICC_IAR bit assignments

Bit	Name	Function
[31:13]	-	Reserved.
[12:10]	CPUID	For SGIs in a multiprocessor implementation, this field identifies the processor that requested the interrupt. It returns the number of the CPU interface that made the request, for example a value of 3 means the request was generated by a write to the GICD_SGIR on CPU interface 3. For all other interrupts this field is RAZ.
[9:0]	Interrupt ID	The interrupt ID.

Figure 4-28 shows the GICC_EOIR bit assignments.

Figure 4-28 GICC_EOIR bit assignments

Table 4-36 shows the GICC_EOIR bit assignments.

Table 4-36 GICC_EOIR bit assignments

Bits	Name	Function
[31:13]	-	Reserved.
[12:10]	CPUID	On a multiprocessor implementation, if the write refers to an SGI, this field contains the CPUID value from the corresponding GICC_IAR access. In all other cases this field SBZ.
[9:0]	EOIINTID	The Interrupt ID value from the corresponding GICC_IAR access.

<Private Timer>

Base address: 0xF8F0_0600

Table 4-1 Timer and watchdog registers

Offset	Type	Reset Value	Function
0x00	RW	0x00000000	*Private Timer Load Register*
0x04	RW	0x00000000	*Private Timer Counter Register*
0x08	RW	0x00000000	*Private Timer Control Register* on page 4-4
0x0C	RW	0x00000000	*Private Timer Interrupt Status Register* on page 4-4

4.2.4 Private Timer Interrupt Status Register

Figure 4-2 on page 4-5 shows the Private Timer Interrupt Status Register bit assignment.

This is a banked register for all Cortex-A9 processors present.

The event flag is a sticky bit that is automatically set when the Counter Register reaches zero. If the timer interrupt is enabled, Interrupt ID 29 is set as pending in the Interrupt Distributor after the event flag is set. The event flag is cleared when written to 1.

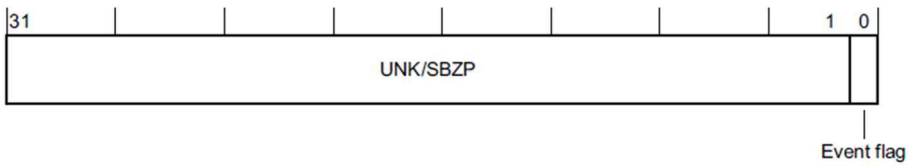


Figure 4-2 Private Timer Interrupt Status Register bit assignment

4. Referring to the following page tables, figure out the address translation unit and mappings from virtual address to physical address. Explain your answer in detail. Refer to the page table entry information in the next page. **(20 points)**

```
// 1st level page table (=csd_MMUTable)

csd_MMUTable:
.set  SECT,  0x100000
.word  SECT + 0x15de6
.set SECT,  SECT+0x400000
.word  csd_MMUTable_lv2 + 0x1e1
.set  SECT, SECT - 0x200000
.word  SECT + 0x15de6


//2nd level page table (=csd_MMUTable_lv2)

csd_MMUTable_lv2:
.word 0xAAAAA002
.word 0xBBBBB002
```

Translation Unit	Virtual address range		Physical address range
		→	

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.

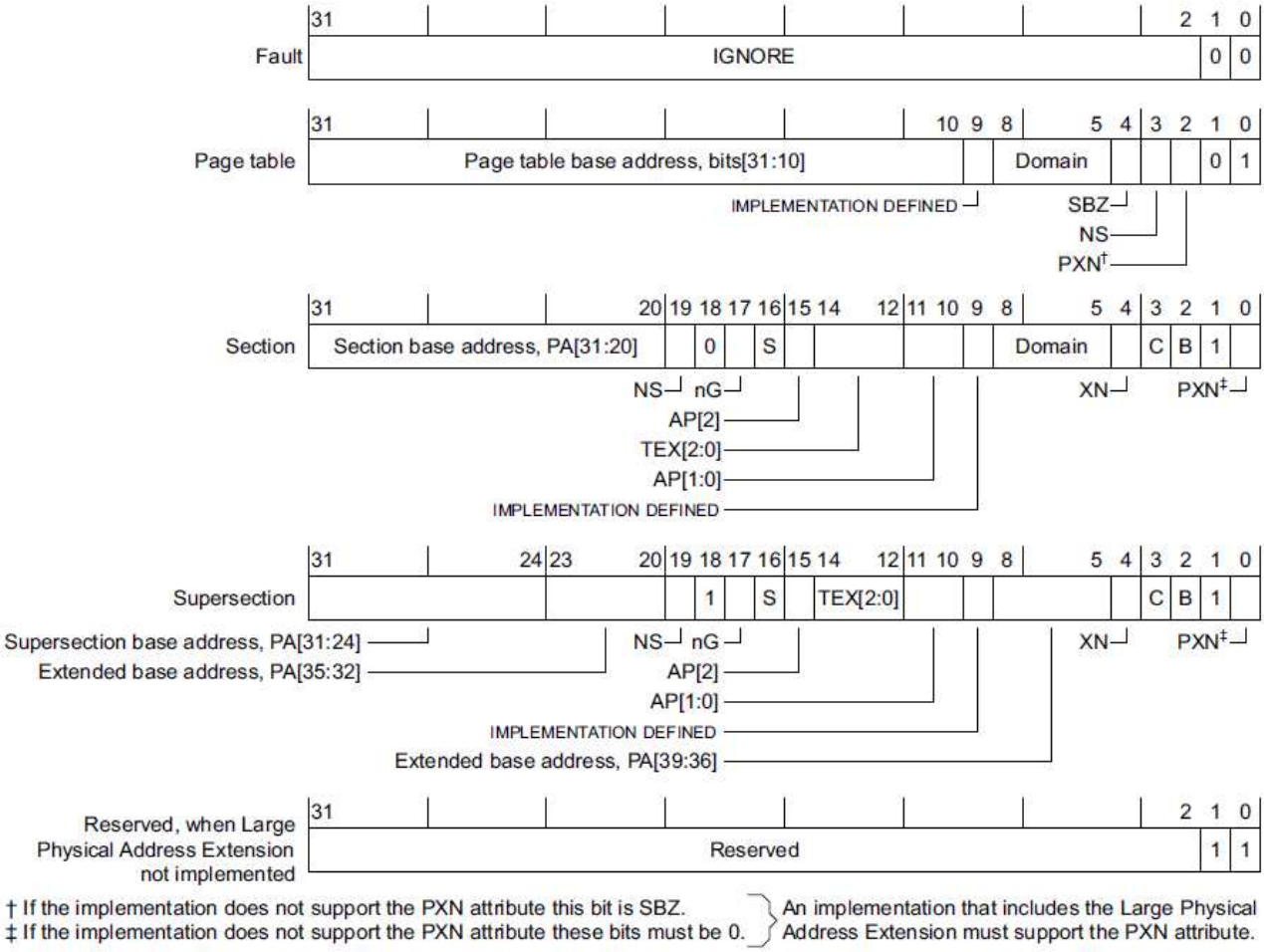


Figure B3-4 Short-descriptor first-level descriptor formats

Short-descriptor translation table second-level descriptor formats

Figure B3-5 shows the possible formats of a second-level descriptor.

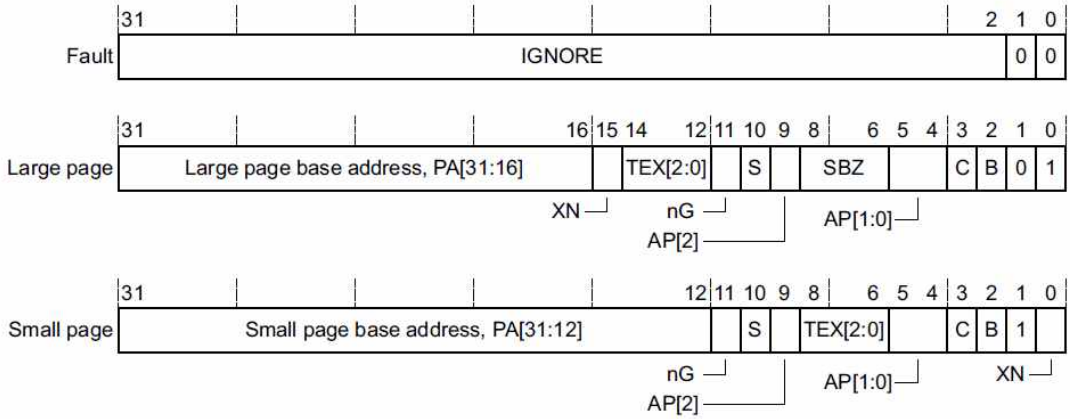


Figure B3-5 Short-descriptor second-level descriptor formats

5. CPU will execute the following 3 instructions. In the worst-case, what could happen in L1 caches? There are separate L1 instruction and data caches. Assume that the cache line size is 4 words (=16 bytes). **Explain** your answer in detail. **(10 points)**

Memory location	Instruction sequence	Worst case scenario in L1 Caches (I$ and D$)
0x018	`sub  r0, r1, r2`	
0x01C	`mov  r10, #0xA004`	
0x020	`ldmfd  sp!, {r0-r12}`	