UNIVERSITY OF DUBLIN
TRINITY COLLEGE

Faculty of Engineering, Mathematics and Science

School of Computer Science & Statistics

BA (Mod.) Computer Science
Senior Freshman Examination

Trinity Term 2011

Microprocessor Systems

Wednesday 4 May 2011 **Goldsmith Hall** **14:00–16:00**

Dr Mike Brady

Instructions to Candidates:
Attempt **two** questions. All questions carry equal marks. Each question is scored out of a total of 20 marks.

You may not start this examination until you are instructed to do so by the Invigilator.

Materials permitted for this examination:
An ASCII code table (one page) and an ARM Instruction Set Summary (two pages) accompany this examination paper.

Non-programmable calculators are permitted for this examination — please indicate the make and model of your calculator on each answer book used.

1. (a) What is an Instruction Set Architecture (ISA)? What is the relationship between an ISA and a processor's Instruction Set? [4 marks]

   (b) Describe the conventional four-stage pipeline. What problems or difficulties occur with pipelining? [4 marks]

   (c) What is meant by superscalar architecture? How does it relate to the sequential execution semantics of a conventional ISA? What obstacles are there to using superscalar architectures? [4 marks]

   (c) List the components and properties of the *memory hierarchy*, and hence explain why a memory hierarchy seems to be necessary in the first place. Describe the function and operation of cache memories. In your answer, discuss the different types of cache organization and replacement polices, and list their advantages and disadvantages. [8 marks]

2. (a) In the context of the ARM processor, what is an *exception*? What happens when an exception occurs? [2 marks]

   (b) What modes of operation does the ARM provide? What are they for? What are the particular features of the FIQ mode? [4 marks]

   (c) Returning from an exception is a little bit tricky on the ARM. What are the difficulties, and how are they dealt with? [4 marks]

   (d) Write a complete ARM IRQ handler to handle an interrupt that occurs exactly every 16.12356 milliseconds (10^{-3} seconds). The task of the interrupt handler is to update an elapsed time counter, organized as three bytes: Hours, Minutes and Seconds, though you can use extra storage if needed. Note that the interrupt handler should not introduce imprecision by making any approximations. [10 marks]

3. (a) Outline the steps needed to develop and test an assembly language program on the experimental ARM boards. [4 marks]

   (b) Describe how to implement a hardware and interrupt-driven software system to display a hexadecimal digit on the seven segment display installed on the breadboard of the ARM experimental board. Let the interrupt be a timer interrupt, occurring at 2 millisecond intervals. Draw a diagram of the setup, detailing the components needed, how they are connected and what their function is. Explain how the program is meant to work, and write the code necessary to initialise and run the system [16 marks]

ASCII Code

Row Number	Column Number 000	001	010	011	100	101	110	111
0000	*NUL*	*DLE*	◊	0	@	P	`	p
0001	*SOH*	*DC1*	!	1	A	Q	a	q
0010	*STX*	*DC2*	"	2	B	R	b	r
0011	*ETX*	*DC3*	#	3	C	S	c	s
0100	*EOT*	*DC4*	$	4	D	T	d	t
0101	*ENQ*	*NAK*	%	5	E	U	e	u
0110	*ACK*	*SYN*	&	6	F	V	f	v
0111	*BELL*	*ETB*	'	7	G	W	g	w
1000	*BS*	*CAN*	(	8	H	X	h	x
1001	*HT*	*EM*	)	9	I	Y	i	y
1010	*LF*	*SUB*	*	:	J	Z	j	z
1011	*VT*	*ESC*	+	;	K	[	k	{
1100	*FF*	*FS*	,	<	L	\	l	|
1101	*CR*	*GS*	-	=	M	]	m	}
1110	*SO*	*RS*	.	>	N	^	n	~
1111	*SI*	*US*	/	?	O	_	o	*DEL*

The ASCII code of a character is found by combining its Column Number (given in 3-bit binary) with its Row Number (given in 4-bit binary).
The Column Number forms bits 6, 5 and 4 of the ASCII, and the Row Number forms bits 3, 2, 1 and 0 of the ASCII.

Example of use: to get ASCII code for letter "n", locate it in Column **110**, Row **1110.** Hence its ASCII code is **1101110**.

The **Control Code** mnemonics are given in italics above; e.g. *CR* for Carriage Return, *LF* for Line Feed, *BELL* for the Bell, *DEL* for Delete.

The Space is ASCII 0100000, and is shown as ◊ here.

ARM7TDMI Instruction Set Summary

Operation	Description	Assembler
Move	Move	MOV{cond}{S} Rd, <Oprnd2>
	Move NOT	MVN{cond}{S} Rd, <Oprnd2>
	Move SPSR to register	MRS{cond} Rd, SPSR
	Move CPSR to register	MRS{cond} Rd, CPSR
	Move register to SPSR	MSR{cond} SPSR{field}, Rm
	Move register to CPSR	MSR{cond} CPSR{field}, Rm
	Move immediate to SPSR flags	MSR{cond} SPSR_f, #32bit_Imm
	Move immediate to CPSR flags	MSR{cond} CPSR_f, #32bit_Imm
Arithmetic	Add	ADD{cond}{S} Rd, Rn, <Oprnd2>
	Add with carry	ADC{cond}{S} Rd, Rn, <Oprnd2>
	Subtract	SUB{cond}{S} Rd, Rn, <Oprnd2>
	Subtract with carry	SBC{cond}{S} Rd, Rn, <Oprnd2>
	Subtract reverse subtract	RSB{cond}{S} Rd, Rn, <Oprnd2>
	Subtract reverse subtract with carry	RSC{cond}{S} Rd, Rn, <Oprnd2>
	Multiply	MUL{cond}{S} Rd, Rm, Rs
	Multiply accumulate	MLA{cond}{S} Rd, Rm, Rs, Rn
	Multiply unsigned long	UMULL{cond}{S} RdLo, RdHi, Rm, Rs
	Multiply unsigned accumulate long	UMLAL{cond}{S} RdLo, RdHi, Rm, Rs
	Multiply signed long	SMULL{cond}{S} RdLo, RdHi, Rm, Rs
	Multiply signed accumulate long	SMLAL{cond}{S} RdLo, RdHi, Rm, Rs
	Compare	CMP{cond} Rd, <Oprnd2>
	Compare negative	CMN{cond} Rd, <Oprnd2>
Logical	Test	TST{cond} Rn, <Oprnd2>
	Test equivalence	TEQ{cond} Rn, <Oprnd2>
	AND	AND{cond}{S} Rd, Rn, <Oprnd2>
	EOR	EOR{cond}{S} Rd, Rn, <Oprnd2>
	ORR	ORR{cond}{S} Rd, Rn, <Oprnd2>
	Bit clear	BIC{cond}{S} Rd, Rn, <Oprnd2>
Branch	Branch	B{cond} label
	Branch with link	BL{cond} label
	Branch and exchange instruction set	BX{cond} Rn
Load	Word	LDR{cond} Rd, <a_mode2>
	Word with user-mode privilege	LDR{cond}T Rd, <a_mode2P>
	Byte	LDR{cond}B Rd, <a_mode2>
	Byte with user-mode privilege	LDR{cond}BT Rd, <a_mode2P>
	Byte signed	LDR{cond}SB Rd, <a_mode3>
	Halfword	LDR{cond}H Rd, <a_mode3>
	Halfword signed	LDR{cond}SH Rd, <a_mode3>
Multiple block data operations	Increment before	LDM{cond}IB Rd{!}, <reglist>{^}
	Increment after	LDM{cond}IA Rd{!}, <reglist>{^}
	Decrement before	LDM{cond}DB Rd{!}, <reglist>{^}
	Decrement after	LDM{cond}DA Rd{!}, <reglist>{^}
	Stack operations	LDM{cond}<a_mode4L> Rd{!}, <reglist>
	Stack operations and restore CPSR	LDM{cond}<a_mode4L> Rd{!}, <reglist+pc>^
	User registers	LDM{cond}<a_mode4L> Rd{!}, <reglist>^
Store	Word	STR{cond} Rd, <a_mode2>
	Word with User-mode privilege	STR{cond}T Rd, <a_mode2P>
	Byte	STR{cond}B Rd, <a_mode2>
	Byte with User-mode privilege	STR{cond}BT Rd, <a_mode2P>
	Halfword	STR{cond}H Rd, <a_mode3>
	Multiple	-
	Block data operations	-
	Increment before	STM{cond}IB Rd{!}, <reglist>{^}
	Increment after	STM{cond}IA Rd{!}, <reglist>{^}
	Decrement before	STM{cond}DB Rd{!}, <reglist>{^}
	Decrement after	STM{cond}DA Rd{!}, <reglist>{^}
	Stack operations	STM{cond}<a_mode4S> Rd{!}, <reglist>
	User registers	STM{cond}<a_mode4S> Rd{!}, <reglist>^
Swap	Word	SWP{cond} Rd, Rm, [Rn]
	Byte	SWP{cond}B Rd, Rm, [Rn]
Coprocessors	Data operations	CDP{cond} p<cpnum>, <op1>, CRd, CRn, CRm, <op2>
	Move to ARM register from coprocessor	MRC{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>
	Move to coprocessor from ARM register	MCR{cond} p<cpnum>, <op1>, Rd, CRn, CRm, <op2>
	Load	LDC{cond} p<cpnum>, CRd, <a_mode5>
	Store	STC{cond} p<cpnum>, CRd, <a_mode5>
Software Interrupt		SWI 24bit_Imm

Source: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0234b/i1010871.html, March 2011

ARM7TDMI Instruction Set Summary

Addressing Mode 2, <a_mode2>

Operation	Assembler
Immediate offset	[Rn, #+/-12bit_Offset]
Register offset	[Rn, +/-Rm]
Scaled register offset	[Rn, +/-Rm, LSL #5bit_shift_imm]
	[Rn, +/-Rm, LSR #5bit_shift_imm]
	[Rn, +/-Rm, ASR #5bit_shift_imm]
	[Rn, +/-Rm, ROR #5bit_shift_imm]
	[Rn, +/-Rm, RRX]
Pre-indexed immediate offset	[Rn, #+/-12bit_Offset]!
Pre-indexed register offset	[Rn, +/-Rm]!
Pre-indexed scaled register offset	[Rn, +/-Rm, LSL #5bit_shift_imm]!
	[Rn, +/-Rm, LSR #5bit_shift_imm]!
	[Rn, +/-Rm, ASR #5bit_shift_imm]!
	[Rn, +/-Rm, ROR #5bit_shift_imm]!
	[Rn, +/-Rm, RRX]!
Post-indexed immediate offset	[Rn], #+/-12bit_Offset
Post-indexed register offset	[Rn], +/-Rm
Post-indexed scaled register offset	[Rn], +/-Rm, LSL #5bit_shift_imm
	[Rn], +/-Rm, LSR #5bit_shift_imm
	[Rn], +/-Rm, ASR #5bit_shift_imm
	[Rn], +/-Rm, ROR #5bit_shift_imm
	[Rn, +/-Rm, RRX]

Addressing Mode 2 (Privileged), <a_mode2P>

Operation	Assembler
Immediate offset	[Rn, #+/-12bit_Offset]
Register offset	[Rn, +/-Rm]
Scaled register offset	[Rn, +/-Rm, LSL #5bit_shift_imm]
	[Rn, +/-Rm, LSR #5bit_shift_imm]
	[Rn, +/-Rm, ASR #5bit_shift_imm]
	[Rn, +/-Rm, ROR #5bit_shift_imm]
	[Rn, +/-Rm, RRX]
Post-indexed immediate offset	[Rn], #+/-12bit_Offset
Post-indexed register offset	[Rn], +/-Rm
Post-indexed scaled register offset	[Rn], +/-Rm, LSL #5bit_shift_imm
	[Rn], +/-Rm, LSR #5bit_shift_imm
	[Rn], +/-Rm, ASR #5bit_shift_imm
	[Rn], +/-Rm, ROR #5bit_shift_imm
	[Rn, +/-Rm, RRX]

Addressing Mode 3, <a_mode3>

Operation	Assembler
Immediate offset	[Rn, #+/-8bit_Offset]
Pre-indexed	[Rn, #+/-8bit_Offset]!
Post-indexed	[Rn], #+/-8bit_Offset
Register	[Rn, +/-Rm]
Pre-indexed	[Rn, +/-Rm]!
Post-indexed	[Rn], +/-Rm

Addressing Mode 4 (Load), <a_mode4L>

Addressing mode	Stack type
IA Increment after	FD Full descending
IB Increment before	ED Empty descending
DA Decrement after	FA Full ascending
DB Decrement before	EA Empty ascending

Addressing Mode 4 (Store), <a_mode4S>

Addressing mode	Stack type
IA Increment after	EA Empty ascending
IB Increment before	FA Full ascending
DA Decrement after	ED Empty descending
DB Decrement before	FD Full descending

Addressing Mode 5 (coprocessor data transfer), <a_mode5>

Operation	Assembler
Immediate offset	[Rn, #+/-(8bit_Offset*4)]
Pre-indexed	[Rn, #+/-(8bit_Offset*4)]!
Post-indexed	[Rn], #+/-(8bit_Offset*4)

Operand 2, <Oprnd2>

Operation	Assembler
Immediate value	#32bit_Imm
Logical shift left	Rm, LSL #5bit_Imm
Logical shift right	Rm, LSR #5bit_Imm
Arithmetic shift right	Rm, ASR #5bit_Imm
Rotate right	Rm, ROR #5bit_Imm
Register	Rm
Logical shift left	Rm, LSL Rs
Logical shift right	Rm, LSR Rs
Arithmetic shift right	Rm, ASR Rs
Rotate right	Rm, ROR Rs
Rotate right extended	Rm, RRX

Fields, {field}

Suffix	Sets
_c	Control field mask bit (bit 3)
_f	Flags field mask bit (bit 0)
_s	Status field mask bit (bit 1)
_x	Extension field mask bit (bit 2)

Condition fields, {cond}

Suffix	Description
EQ	Equal
NE	Not equal
CS	Unsigned higher, or same
CC	Unsigned lower
MI	Negative
PL	Positive, or zero
VS	Overflow
VC	No overflow
HI	Unsigned higher
LS	Unsigned lower, or same
GE	Greater, or equal
LT	Less than
GT	Greater than
LE	Less than, or equal
AL	Always

Source: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0234b/i1010871.html, March 2011