装订线内

不要答题

北京大学信息科学技术学院考试试卷

考试科目：计算机组织与体系结构 **姓名：**_____ **学号：**_____

考试时间：2020 年 6 月 10 日 **任课教师：**陆俊林

题号	一	二	三	四	五		总分
分数							
阅卷人							

北京大学考场纪律

1、考生进入考场后，按照监考老师安排隔位就座，将学生证放在桌面上。无学生证者不能参加考试；迟到超过 15 分钟不得入场。在考试开始 30 分钟后方可交卷出场。

2、除必要的文具和主考教师允许的工具书、参考书、计算器以外，其它所有物品（包括空白纸张、手机、或有存储、编程、查询功能的电子用品等）不得带入座位，已经带入考场的必须放在监考人员指定的位置。

3、考试使用的试题、答卷、草稿纸由监考人员统一发放，考试结束时收回，一律不准带出考场。若有试题印制问题请向监考教师提出，不得向其他考生询问。提前答完试卷，应举手示意请监考人员收卷后方可离开；交卷后不得在考场内逗留或在附近高声交谈。未交卷擅自离开考场，不得重新进入考场答卷。考试结束时间到，考生立即停止答卷，在座位上等待监考人员收卷清点后，方可离场。

4、考生要严格遵守考场规则，在规定时间内独立完成答卷。不准交头接耳，不准偷看、夹带、抄袭或者有意让他人抄袭答题内容，不准接传答案或者试卷等。凡有违纪作弊者，一经发现，当场取消其考试资格，并根据《北京大学本科考试工作与学术规范条例》及相关规定严肃处理。

5、考生须确认自己填写的个人信息真实、准确，并承担信息填写错误带来的一切责任与后果。

学校倡议所有考生以北京大学学生的荣誉与诚信答卷，共同维护北京大学的学术声誉。

以下为试题和答题纸，共 11 页。

得分

第一题 填空题（共 20 分）

1. 计算机执行指令的基本步骤按顺序是：取指、 译码 、 执行 、写回。

2. 通常认为第一台存储程序式的电子计算机是 EDSAC 。

3. Intel 4004 的字长为 8 位，首款 MIPS CPU（R2000）的字长为 32 位。

4. 以课程中讲解的 4-bit 第一版乘法器为例，对于二进制数 0010×0011，当第 2 次循环结束时，被乘数寄存器、乘数寄存器和乘积寄存器的值分别是 00001000 , 0000 , 00000110。（直接写二进制数）

5. 以课程中讲解的 4-bit 第一版除法器为例，对于二进制数 0111÷0010，当第 2 次循环结束时，商寄存器、除数寄存器和余数寄存器的值是 0000 , 00001000 , 00000111 。（直接写二进制数）

6. 在 MIPS 的五级流水线结构中，计算访存的地址是在 执行 阶段完成的。在 Intel 486 的五级流水线结构中，计算访存的地址是在 地址生成 阶段完成的。

7. 通常 CMOS 集成电路中，或门由一个 或非 门和一个 非 门构成。

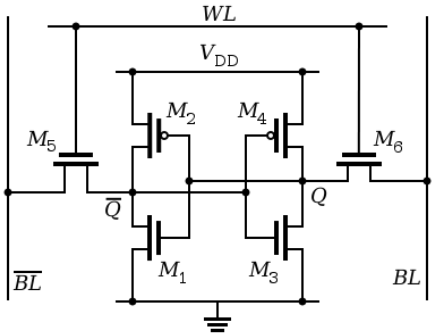
8、64-bit 数据宽度的 DDR3-1600 SDRAM 的峰值带宽为 12.8GB/s，接口时钟频率为 800MHz，芯片内部采用了 8 位数据预取技术，其核心频率是 200 MHz。

9、一个容量为 1GB 的内存，如果不考虑冗余空间，那可以存储 2^33 个二进制位的信息；一个传输率为 100Mbps 的内存接口，每秒钟最多能输出 10^8 个二进制位的信息。（填写十进制数）

10、若向一个 SRAM 单元写入 1，则需置 BL=1， $\overline{BL}$ =0，WL=1。请问，此时 SRAM 结构图中的晶体管 M_1~M_6 分别处于连通还是关闭状态？在下面空格处填写“连通”或“关闭”。

M_1：连通　　M_2：关闭　　M_3：关闭

M_4：连通　　M_5：连通　　M_6：连通



得分

第二题（20 分）

现有一段 MIPS 机器代码，见下表 B 列；其汇编语言源代码已残缺不全，见下表 A 列。请以 A 列为参考，手工反汇编 B 列，填写在 C 列。C 列中标星号（*）的空格必须填写。其它空格不计分，可不填，但对回答本大题中后续小题有作用。

	汇编语言代码（A 列）	机器语言代码（B 列）	反汇编代码（C 列）
1.	l[redacted] 0x1[redacted]	3c011001	lui $at,0x1001
2.	[redacted]	342e0040	
3.	[redacted] $t7, 1	adc00000	*sw $zero, 0x0($t6)
4.	sw $t[redacted]	240f0001	*addiu $t7, $zero, 0x1
5.	[redacted] $t3, 0	adcf0004	
6.	[redacted]	240b0000	
7.	[redacted] $t5, 8	240c0004	
8.	[redacted]	240d0008	
9.	[redacted]	2418002c	
10.	LOOP:	24190014	*addiu $t9, $zero ,0x14
11.	[redacted], $t3	01cb7820	*add $t7, $t6, $t3
12.	[redacted]	8de80000	*lw $t0, 0x0($t7)
13.	add $t7, $t6, $t4	01cc7820	
14.	lw [redacted]	8de90000	
15.	[redacted] $t6, $t5	01cd7820	
16.	[redacted]	01095020	*add $t2, $t0, $t1
17.	[redacted]	adea0000	*sw $t2, 0x0($t7)
18.	[redacted]	000c5821	*addu $t3, $zero, $t2
19.	move [redacted]	000d6021	
20.	[redacted]	21ad0004	
21.	[redacted] EXIT	01b8082a	*slt $at, $t5, $t8
22.	b[redacted]	10200002	*beq $at, $zero, EXIT
23.	EXIT:	0159082a	*slt $at, $t2, $t9
24.		1420fff2	*bne $at, $zero, LOOP

2．请问 A 列第 1 行的代码应该是什么？

答： lui $at,0x1001

3、这段代码最有可能的功能是什么？

答：循环复制数组

参考材料

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

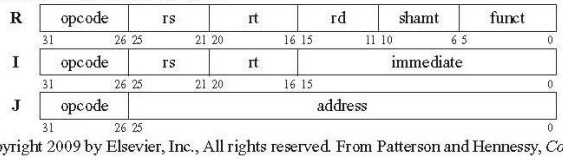
MIPS Reference Data ①

CORE INSTRUCTION SET

NAME, MNEMONIC		FOR-MAT	OPERATION (in Verilog)		OPCODE / FUNCT (Hex)
Add	add	R	R[rd] = R[rs] + R[rt]	(1)	0 / 20_{hex}
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm	(1,2)	8_{hex}
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm	(2)	9_{hex}
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]		0 / 21_{hex}
And	and	R	R[rd] = R[rs] & R[rt]		0 / 24_{hex}
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm	(3)	c_{hex}
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4_{hex}
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5_{hex}
Jump	j	J	PC=JumpAddr	(5)	2_{hex}
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr	(5)	3_{hex}
Jump Register	jr	R	PC=R[rs]		0 / 08_{hex}
Load Byte Unsigned	lbu	I	R[rt]={24'b0,M[R[rs]+SignExtImm](7:0)}	(2)	24_{hex}
Load Halfword Unsigned	lhu	I	R[rt]={16'b0,M[R[rs]+SignExtImm](15:0)}	(2)	25_{hex}
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]	(2,7)	30_{hex}
Load Upper Imm.	lui	I	R[rt] = {imm, 16'b0}		f_{hex}
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]	(2)	23_{hex}
Nor	nor	R	R[rd] = ~ (R[rs] | R[rt])		0 / 27_{hex}
Or	or	R	R[rd] = R[rs] | R[rt]		0 / 25_{hex}
Or Immediate	ori	I	R[rt] = R[rs] | ZeroExtImm	(3)	d_{hex}
Set Less Than	slt	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0		0 / $2a_{hex}$
Set Less Than Imm.	slti	I	R[rt] = (R[rs] < SignExtImm)? 1 : 0	(2)	a_{hex}
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] < SignExtImm) ? 1 : 0	(2,6)	b_{hex}
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] < R[rt]) ? 1 : 0	(6)	0 / $2b_{hex}$
Shift Left Logical	sll	R	R[rd] = R[rt] << shamt		0 / 00_{hex}
Shift Right Logical	srl	R	R[rd] = R[rt] >> shamt		0 / 02_{hex}
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)	(2)	28_{hex}
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (*atomic*) ? 1 : 0	(2,7)	38_{hex}
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)	(2)	29_{hex}
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]	(2)	$2b_{hex}$
Subtract	sub	R	R[rd] = R[rs] - R[rt]	(1)	0 / 22_{hex}
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]		0 / 23_{hex}

(1) May cause overflow exception
(2) SignExtImm = { 16{immediate[15]}, immediate }
(3) ZeroExtImm = { 16{1b'0}, immediate }
(4) BranchAddr = { 14{immediate[15]}, immediate, 2'b0 }
(5) JumpAddr = { PC+4[31:28], address, 2'b0 }
(6) Operands considered unsigned numbers (vs. 2's comp.)
(7) Atomic test&set pair; R[rt] = 1 if pair atomic, 0 if not atomic

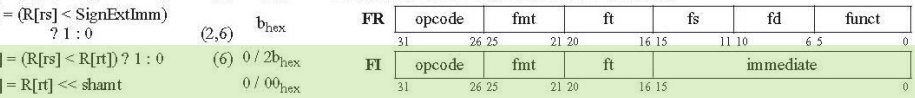
BASIC INSTRUCTION FORMATS



ARITHMETIC CORE INSTRUCTION SET ②

NAME, MNEMONIC		FOR-MAT	OPERATION		OPCODE / FMT /FT / FUNCT (Hex)
Branch On FP True	bc1t	FI	if(FPcond)PC=PC+4+BranchAddr	(4)	11/8/1/--
Branch On FP False	bc1f	FI	if(!FPcond)PC=PC+4+BranchAddr	(4)	11/8/0/--
Divide	div	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]		0/--/--/1a
Divide Unsigned	divu	R	Lo=R[rs]/R[rt]; Hi=R[rs]%R[rt]	(6)	0/--/--/1b
FP Add Single	add.s	FR	F[fd]= F[fs] + F[ft]		11/10/--/0
FP Add Double	add.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} + {F[ft],F[ft+1]}		11/11/--/0
FP Compare Single	c.x.s*	FR	FPcond = (F[fs] *op* F[ft]) ? 1 : 0		11/10/--/*y*
FP Compare Double	c.x.d*	FR	FPcond = ({F[fs],F[fs+1]} *op* {F[ft],F[ft+1]}) ? 1 : 0		11/11/--/*y*
* (*x* is eq, lt, or le) (*op* is ==, <, or <=) (*y* is 32, 3c, or 3e)					
FP Divide Single	div.s	FR	F[fd] = F[fs] / F[ft]		11/10/--/3
FP Divide Double	div.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} / {F[ft],F[ft+1]}		11/11/--/3
FP Multiply Single	mul.s	FR	F[fd] = F[fs] * F[ft]		11/10/--/2
FP Multiply Double	mul.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} * {F[ft],F[ft+1]}		11/11/--/2
FP Subtract Single	sub.s	FR	F[fd]=F[fs] - F[ft]		11/10/--/1
FP Subtract Double	sub.d	FR	{F[fd],F[fd+1]} = {F[fs],F[fs+1]} - {F[ft],F[ft+1]}		11/11/--/1
Load FP Single	lwc1	I	F[rt]=M[R[rs]+SignExtImm]	(2)	31/--/--/--
Load FP Double	ldc1	I	F[rt]=M[R[rs]+SignExtImm]; F[rt+1]=M[R[rs]+SignExtImm+4]	(2)	35/--/--/--
Move From Hi	mfhi	R	R[rd] = Hi		0/--/--/10
Move From Lo	mflo	R	R[rd] = Lo		0/--/--/12
Move From Control	mfc0	R	R[rd] = CR[rs]		10/0/--/0
Multiply	mult	R	{Hi,Lo} = R[rs] * R[rt]		0/--/--/18
Multiply Unsigned	multu	R	{Hi,Lo} = R[rs] * R[rt]	(6)	0/--/--/19
Shift Right Arith.	sra	R	R[rd] = R[rt] >>> shamt		0/--/--/3
Store FP Single	swc1	I	M[R[rs]+SignExtImm] = F[rt]	(2)	39/--/--/--
Store FP Double	sdc1	I	M[R[rs]+SignExtImm] = F[rt]; M[R[rs]+SignExtImm+4] = F[rt+1]	(2)	3d/--/--/--

FLOATING-POINT INSTRUCTION FORMATS



PSEUDOINSTRUCTION SET

NAME	MNEMONIC	OPERATION
Branch Less Than	blt	if(R[rs]<R[rt]) PC = Label
Branch Greater Than	bgt	if(R[rs]>R[rt]) PC = Label
Branch Less Than or Equal	ble	if(R[rs]<=R[rt]) PC = Label
Branch Greater Than or Equal	bge	if(R[rs]>=R[rt]) PC = Label
Load Immediate	li	R[rd] = immediate
Move	move	R[rd] = R[rs]

REGISTER NAME, NUMBER, USE, CALL CONVENTION

NAME	NUMBER	USE	PRESERVED ACROSS A CALL?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0-$v1	2-3	Values for Function Results and Expression Evaluation	No
$a0-$a3	4-7	Arguments	No
$t0-$t7	8-15	Temporaries	No
$s0-$s7	16-23	Saved Temporaries	Yes
$t8-$t9	24-25	Temporaries	No
$k0-$k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

OPCODES, BASE CONVERSION, ASCII SYMBOLS ③

MIPS opcode (31:26)	(1) MIPS funct (5:0)	(2) MIPS funct (5:0)	Binary	Decimal	Hexadecimal	ASCII Character	Decimal	Hexadecimal	ASCII Character
(1)	sll	add.*f*	00 0000	0	0	NUL	64	40	@
		sub.*f*	00 0001	1	1	SOH	65	41	A
j	srl	mul.*f*	00 0010	2	2	STX	66	42	B
jal	sra	div.*f*	00 0011	3	3	ETX	67	43	C
beq	sllv	sqrt.*f*	00 0100	4	4	EOT	68	44	D
bne		abs.*f*	00 0101	5	5	ENQ	69	45	E
blez	srlv	mov.*f*	00 0110	6	6	ACK	70	46	F
bgtz	srav	neg.*f*	00 0111	7	7	BEL	71	47	G
addi	jr		00 1000	8	8	BS	72	48	H
addiu	jalr		00 1001	9	9	HT	73	49	I
slti	movz		00 1010	10	a	LF	74	4a	J
sltiu	movn		00 1011	11	b	VT	75	4b	K
andi	syscall	round.w.*f*	00 1100	12	c	FF	76	4c	L
ori	break	trunc.w.*f*	00 1101	13	d	CR	77	4d	M
xori		ceil.w.*f*	00 1110	14	e	SO	78	4e	N
lui	sync	floor.w.*f*	00 1111	15	f	SI	79	4f	O
	mfhi		01 0000	16	10	DLE	80	50	P
(2)	mthi		01 0001	17	11	DC1	81	51	Q
	mflo	movz.*f*	01 0010	18	12	DC2	82	52	R
	mtlo	movn.*f*	01 0011	19	13	DC3	83	53	S
			01 0100	20	14	DC4	84	54	T
			01 0101	21	15	NAK	85	55	U
			01 0110	22	16	SYN	86	56	V
			01 0111	23	17	ETB	87	57	W
	mult		01 1000	24	18	CAN	88	58	X
	multu		01 1001	25	19	EM	89	59	Y
	div		01 1010	26	1a	SUB	90	5a	Z
	divu		01 1011	27	1b	ESC	91	5b	[
			01 1100	28	1c	FS	92	5c	\
			01 1101	29	1d	GS	93	5d	]
			01 1110	30	1e	RS	94	5e	^
			01 1111	31	1f	US	95	5f	_
lb	add	cvt.s.*f*	10 0000	32	20	Space	96	60	`
lh	addu	cvt.d.*f*	10 0001	33	21	!	97	61	a
lwl	sub		10 0010	34	22	"	98	62	b
lw	subu		10 0011	35	23	#	99	63	c
lbu	and	cvt.w.*f*	10 0100	36	24	$	100	64	d
lhu	or		10 0101	37	25	%	101	65	e
lwr	xor		10 0110	38	26	&	102	66	f
	nor		10 0111	39	27	'	103	67	g
sb			10 1000	40	28	(	104	68	h
sh			10 1001	41	29	)	105	69	i
swl	slt		10 1010	42	2a	*	106	6a	j
sw	sltu		10 1011	43	2b	+	107	6b	k
			10 1100	44	2c	,	108	6c	l
			10 1101	45	2d	-	109	6d	m
swr			10 1110	46	2e	.	110	6e	n
cache			10 1111	47	2f	/	111	6f	o
ll	tge	c.f.*f*	11 0000	48	30	0	112	70	p
lwc1	tgeu	c.un.*f*	11 0001	49	31	1	113	71	q
lwc2	tlt	c.eq.*f*	11 0010	50	32	2	114	72	r
pref	tltu	c.ueq.*f*	11 0011	51	33	3	115	73	s
	teq	c.olt.*f*	11 0100	52	34	4	116	74	t
ldc1		c.ult.*f*	11 0101	53	35	5	117	75	u
ldc2	tne	c.ole.*f*	11 0110	54	36	6	118	76	v
		c.ule.*f*	11 0111	55	37	7	119	77	w
sc		c.sf.*f*	11 1000	56	38	8	120	78	x
swc1		c.ngle.*f*	11 1001	57	39	9	121	79	y
swc2		c.seq.*f*	11 1010	58	3a	:	122	7a	z
		c.ngl.*f*	11 1011	59	3b	;	123	7b	{
		c.lt.*f*	11 1100	60	3c	<	124	7c	|
sdc1		c.nge.*f*	11 1101	61	3d	=	125	7d	}
sdc2		c.le.*f*	11 1110	62	3e	>	126	7e	~
		c.ngt.*f*	11 1111	63	3f	?	127	7f	DEL

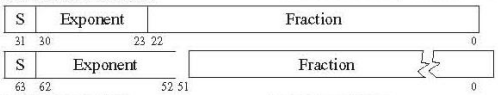
(1) opcode(31:26) == 0
(2) opcode(31:26) == 17_{ten} (11_{hex}); if fmt(25:21)==16_{ten} (10_{hex}) *f* = s (single); if fmt(25:21)==17_{ten} (11_{hex}) *f* = d (double)

IEEE 754 FLOATING-POINT STANDARD

$$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent} - \text{Bias})}$$

where Single Precision Bias = 127, Double Precision Bias = 1023.

IEEE Single Precision and Double Precision Formats:

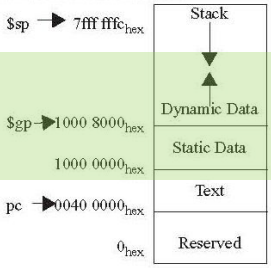


IEEE 754 Symbols ④

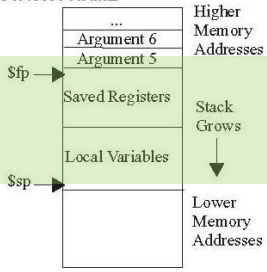
Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

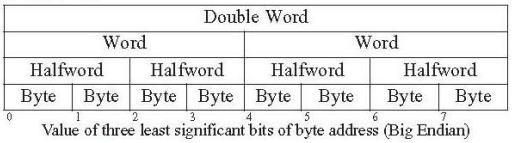
MEMORY ALLOCATION



STACK FRAME

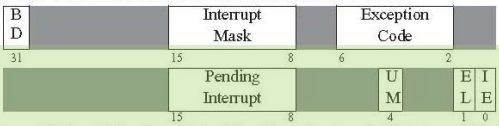


DATA ALIGNMENT



Value of three least significant bits of byte address (Big Endian)

EXCEPTION CONTROL REGISTERS: CAUSE AND STATUS



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception

SIZE PREFIXES (10^x for Disk, Communication; 2^x for Memory)

SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX	SIZE	PREFIX
10^3, 2^{10}	Kilo-	10^{15}, 2^{50}	Peta-	10^{-3}	milli-	10^{-15}	femto-
10^6, 2^{20}	Mega-	10^{18}, 2^{60}	Exa-	10^{-6}	micro-	10^{-18}	atto-
10^9, 2^{30}	Giga-	10^{21}, 2^{70}	Zetta-	10^{-9}	nano-	10^{-21}	zepto-
10^{12}, 2^{40}	Tera-	10^{24}, 2^{80}	Yotta-	10^{-12}	pico-	10^{-24}	yocto-

The symbol for each prefix is just its first letter, except μ is used for micro.

MIPS Reference Data Card ("Green Card") 1. Pull along perforation to separate card 2. Fold bottom side (columns 3 and 4) together

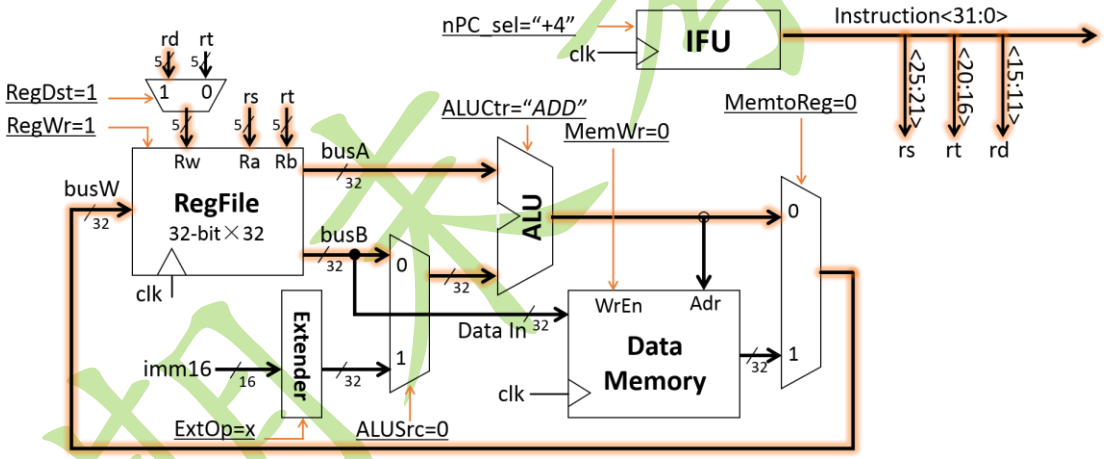
得分	

第三题（20 分）

在 MIPS 指令系统基础上简化而成的 MIPS-Lite 指令系统包含如下指令：

- 无符号加法和减法
  addu rd,rs,rt
  subu rd,rs,rt
- 立即数的逻辑或
  ori rt,rs,imm16
- 装载和存储一个字（Word）
  lw rt,rs,imm16
  sw rt,rs,imm16
- 条件分支
  beq rs,rt,imm16

下图为支持该指令系统的单周期处理器的控制信号和数据通路图，并以 addu 指令为例标出了控制信号的值，请据此回答如下问题。



1. 请分析指令所需的控制信号，将下表填写完整。（注：多位信号的高位在左边）

	addu	subu	ori	lw	sw	beq
RegDst	1	1	0	0	x	x
ALUSrc	0	0	1	1	1	0
MemtoReg	0	0	0	1	x	x
RegWr	1	1	1	1	0	0
MemWr	0	0	0	0	1	0
nPC_sel	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x
ALUctr<1:0>	00(ADD)	01(SUB)	10(OR)	00(ADD)	00(ADD)	01(SUB)

2、如果要将这个单周期处理器改造成 5 级流水线处理器，流水级分别为取指（I）、译码（D）、执行（E）、访存（M）和回写（W）。那在流水线处理器上会出现哪几种“冒险”（harzard）？对于每种冒险，请举出具体指令实例进行说明，然后说明需要如何增加或者修改哪些部件予以解决？

答：

结构冒险：

图中指令与数据存储器分开，不会发生冒险。但可能前一条指令写寄存器 addu $t1,$t2,$t3 位于回写阶段，隔两条指令后一条指令 addu $t5,$t1,$t2 读寄存器位于译码阶段，从而同时发生读写形成结构冒险，需要设置访存的前半周期为写寄存器，译码的后半周读寄存器来避免冒险。

数据冒险：

前一条指令是 addu $t1,$t2,$t3，后一条（或者再后一条）指令是 addu $t3,$t1,$t2，则由于前一条指令仍处于执行（或访存）阶段还未写入寄存器，下一条指令已经位于译码阶段需要读寄存器，从而形成冒险。需要增加从 E 和 M（或 M 和 W）间的寄存器连回 ALU 的旁路来解决该问题。

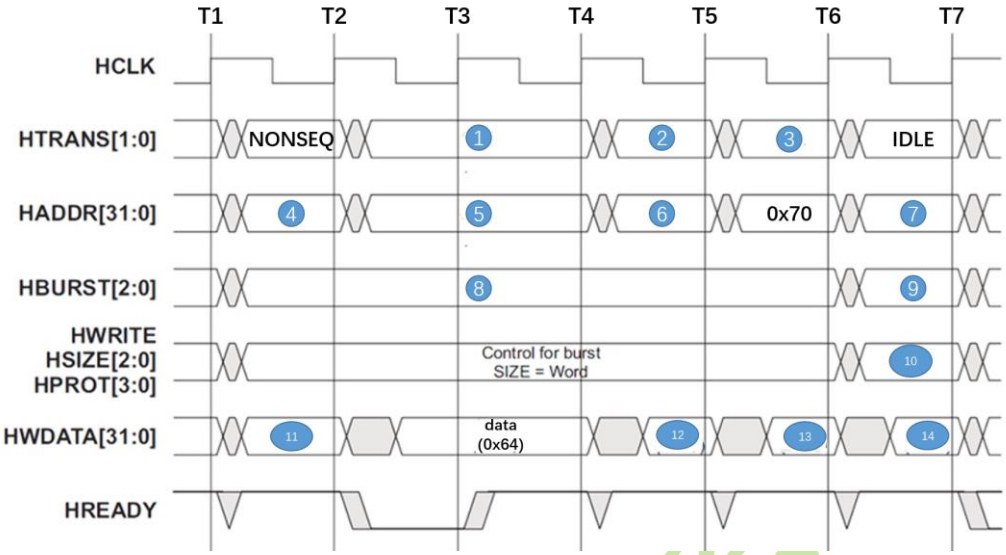
控制冒险：

前一条指令是 beq $t1,$t2,LOOP 后一条指令是 addu $t1,$t2,$t3，则由于前一条指令可能条件满足需要跳转，然而此时下一条指令已经进入流水线从而发生冒险。需要在流水线中加入判断条件指令加入 bubble 的部件。

得分	

第四题（20 分）

这是是一段 AMBA AHB 总线时序图，请回答下列问题。



1、根据图中的提示，补全所有带数字标号的位置的信号值，填到下表中。如果认为是无关或无法确定的值，标记为“X”。

1	SEQ	2	SEQ	3	SEQ		
4	0x64	5	0x68	6	0x6C	7	X
8	INCR4	9	X				
10	X						
11	X	12	Data(0x68)	13	Data(0x6C)	14	Data(0x70)

2、时钟上升沿 T3 时刻，HREADY 信号为低电平。从总线协议上看，这代表什么含义？这样的信号设置，常用于什么场景？

答：

从设备尚未准备好，让主设备维持信号一周期。常用于内存控制器向 CPU 返回尚未准备好数据，还在行选/列选等过程中。

3、上图描绘的这个总线传输，是 AHB 总线诸多类型的传输中使用非常广泛的一种，请描述该类型传输的主要特点和用途。

答：

该类型是 BURST 连发传输

特点是支持大量数据的连续传输，还有递增 INCR 和回卷 WARP 两种方式。

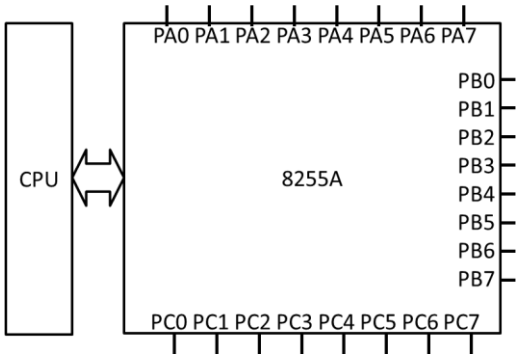
用途可用于外设和主存之间的传输或者主存内部数据大量传输。

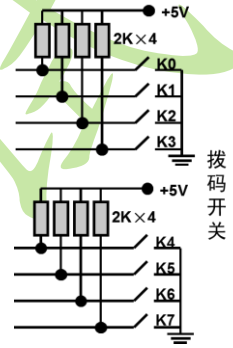
得分

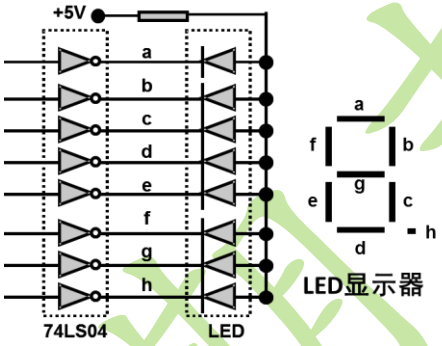
第五题（20 分）

有人使用带并行接口芯片8255A的计算机系统设计了一个方案，完成如下功能：使用8个拨码开关（K7~K0）输入二进制数，两个LED显示管上显示对应的十六进制数。例如，拨码开关（K7~K0）输入为“10100111”，则LED显示管上显示“A7”。输入改变后，输出随之改变，并可以反复输入。

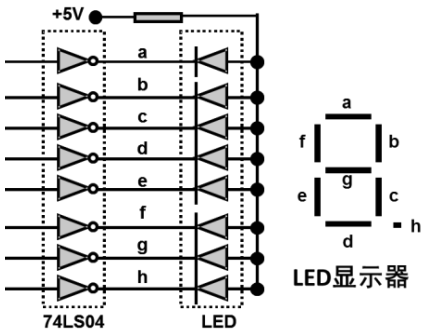
下图中的外设器件与8255A的连接已有其他人去完成，拨码开关连接到8255A的端口A，低位和高位的LED灯分别连接到端口B和端口C。现在由你按后续的要求完成驱动程序代码。







左侧 LED



右侧 LED

补全下面的 x86 汇编程序，实现该设计方案。程序应有完整框架（必要的数据定义和段声明等），包含 8255A 的初始化代码、运行控制代码等，可以完成题目要求的全部功能。要求对自行填写的每行代码加注释说明，并回答已有代码注释中的问题。

```
1)  DATA  SEGMENT
2)  SSEGCODE DB  3FH, 06H, 5BH, 4FH, 66H, 6DH, 7DH, 07H
3)           DB  7FH, 67H, 77H, 7CH, 39H, 5EH, 79H, 71H
4)
5)      ASSUME  CS: CODE, DS: DATA
6)      START:  MOV   AX, DATA
7)              MOV   DS, AX
8)              MOV   AL, 90H
9)              OUT   63H, AL          ; 设置控制字为均按方式 0，端口 A 输入，BC 输出
10)     RDPORTA:
11)             IN    AX, 60H
12)             AND   AL, 0FH
13)
14)             MOV   BX, OFFSET SSGCODE
15)             XLAT                   ; 查表，AL←(BX+AL)
16)             OUT   61H, AL          ; 从 B 输出
17)             MOV   AX, XXXXH
18)     DELAY:  DEC   AX
19)             JNZ   DELAY            ; 循环延时
20)             MOV   AH, 1            ; 问题 1：这里向 AH 中写入数据，什么地方会用到？
21)             INT   16H              ; 问题 2 和 3：INT 是什么指令？会产生什么操作？
22)             JZ    RDPORTA          ; 问题 4 和 5：什么条件下会发生转移？条件码在哪里设置？
23)             MOV  AH, 4CH
24)             INT   21H
25) CODE  ENDS
26)     END  START
```

答：

问题 1：

中断控制程序会判断是否有按键按下

问题 2：

中断指令

问题 3：

INT x 会产生 x 类型中断，进入对应中断控制程序

问题 4：

无按键按下

问题 5：

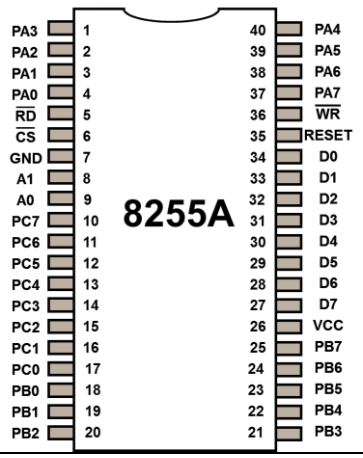
在中断控制程序中设置

附加材料

I/O 端口地址分配

地址空间	器件/接口适配器	实际使用端口
0060~007FH	并行接口片 8255A	0060~0063H

并行通信接口 8255A



重要引脚信号说明

- 片选控制：CS；读控制：RD；写控制：WR
- 端口 A：PA7~PA0；端口 B：PB7~PB0

均为 8 位的端口，但端口 A 的功能更为丰富，可分别设定为输入端口或输出端口

- 端口 C：PC7~PC0

分成两个 4 位的端口，可分别设定为输入端口或输出端口；也可作为端口 A 和端口 B 的“握手”信号

- 复位：RESET；地址：A1、A0；数据：D7~D0

内部端口

A_1	A_0	端口
0	0	端口 A
0	1	端口 B
1	0	端口 C
1	1	控制端口

状态字

控制字