

THE UNIVERSITY OF NEW MEXICO

Midterm Exam – Tuesday, March 23, 9:30am–10:45am, 2010

COMPUTER SCIENCE
CS 341 – Computer Organization
(Time allowed: 75 minutes)

LASTNAME: _____
FIRSTNAME: _____
STUDENT ID NUMBER: _____
SECTION LEADER NAME: _____

Answer all questions in the space provided. Write clearly and legibly, you will not get credit for illegible or incomprehensible answers. This is a closed book, closed notes exam. However, you are allowed to bring the "Green Sheet" from the text book, and a standard calculator. No other aids are allowed. Print your name at the top of every page.

①		25
②		25
③		25
④		25
Total		100

Student Name: _____

CS 341

This page intentionally left mostly blank. . .



① Multiple Choice

[25 points]

*Note. Correct answer = 1p, No answer = 0p, Wrong answer = -1p
The total can not be less than 0.*

True - False questions:

Big and Little Endian concerns only negative numbers	True <input type="checkbox"/>	False <input type="checkbox"/>
<code>.align 2</code> aligns the next datum on a byte boundary	True <input type="checkbox"/>	False <input type="checkbox"/>
\$s-registers must be preserved across subroutine calls	True <input type="checkbox"/>	False <input type="checkbox"/>
With 2's complement addition you can't overflow	True <input type="checkbox"/>	False <input type="checkbox"/>
All MIPS instructions are the same length	True <input type="checkbox"/>	False <input type="checkbox"/>
A branch-and-link instruction updates \$sp	True <input type="checkbox"/>	False <input type="checkbox"/>
The <code>sw</code> instruction stores a half-word in a register	True <input type="checkbox"/>	False <input type="checkbox"/>
<code>lw \$t0, label</code> loads the value stored at <code>label</code> into <code>\$t0</code>	True <input type="checkbox"/>	False <input type="checkbox"/>
An offset is a distance from an address	True <input type="checkbox"/>	False <input type="checkbox"/>
<code>lw \$t0, 16(\$t3)</code> loads the 5th word of an int array to <code>\$t0</code>	True <input type="checkbox"/>	False <input type="checkbox"/>
"Popping" the stack means increasing the stack pointers value	True <input type="checkbox"/>	False <input type="checkbox"/>
A parameter passed on the stack must be an address	True <input type="checkbox"/>	False <input type="checkbox"/>
All load and branch instructions are followed by a delay slot	True <input type="checkbox"/>	False <input type="checkbox"/>
The R4600 CPU can do without a \$fp since it can be calculated	True <input type="checkbox"/>	False <input type="checkbox"/>
In a multi-stage data path, all instructions take the same time to finish	True <input type="checkbox"/>	False <input type="checkbox"/>
An <code>sb</code> instruction is followed by a delay slot	True <input type="checkbox"/>	False <input type="checkbox"/>
All MIPS instructions take one clock cycle to execute	True <input type="checkbox"/>	False <input type="checkbox"/>

Other types of questions:

Memory allocation scheme considered less complex?	Static <input type="checkbox"/>	Dynamic <input type="checkbox"/>
Is used to choose one of several inputs	Multiplexer <input type="checkbox"/>	Demultiplexer <input type="checkbox"/>
Bits usually used to represent a half-word?	16 <input type="checkbox"/>	8 <input type="checkbox"/>
Directive used to tell the debugger the size of current activation record	<code>.frame</code> <input type="checkbox"/>	<code>.mask</code> <input type="checkbox"/>
Register usually stored on stack in a subroutine?	<code>\$sp</code> <input type="checkbox"/>	<code>\$ra</code> <input type="checkbox"/>
Instruction used to call a subroutine	<code>b</code> <input type="checkbox"/>	<code>jal</code> <input type="checkbox"/>
Is a pseudo-instruction	<code>mul</code> <input type="checkbox"/>	<code>addiu</code> <input type="checkbox"/>
Number of bytes that the stack frame size should be evenly divisible by	4 <input type="checkbox"/>	8 <input type="checkbox"/>

Student Name: _____

CS 341

② **Conversions**

[25 points]

For the following questions there are usually a simple double-check that you can do, please also include this double check in the answer so that you can ensure yourself that it is right.

a) Convert the binary number 10111001_2 to decimal. Show your work. [5 points]

b) Convert 15763_8 to decimal. Show your work. [5 points]

c) Convert -101_{10} into 2's complement representation. Show your work. [5 points]

d) Convert 4321_{10} to binary. Show your work. [5 points]

e) Convert 15763_8 to hexademical representation. Show your work. [5 points]

③ Assembly Programming

[25 points]

a) Derive the truth table for A and (B xor C)

[3 points]



C code to generate the truth table that you just generated above might look something like this:

```
#include <stdio.h>
int main(int argc, char **argv) {
    int a, b, c;
    for ( a = 0; a <= 1; a++ )
        for ( b = 0; b <= 1; b++ )
            for ( c = 0; c <= 1; c++ )
                printf ( "%d%d%d %d\n", a, b, c, a & (b ^ c) );

    // ... More code to follow here ...
    return 0;
}
```

b) If we are storing all local variables on the stack, like for the decode function that you are currently working on as an assignment, what should the stack frame for `main` look like? Note, you must assume that the main program contains more code, possibly using the `argc` and `argv` variables, but you can safely assume that no function called from `main` will have more arguments than the call to `printf` that's already there and that no more local variables are declared later in the subroutine. *Please draw a stack diagram that shows the layout of your stack frame!* [6 points]



Student Name: _____

CS 341

c) Based on your (hopefully) correct stack layout from part b), what should the `.mask` directive for the debugger look like? Please motivate your answer. [4 points]

d) Assuming that your prologue and epilogue have been correctly written. Please write the assembly code that implements the C program listed above and prints out the truth table for the boolean expression in part a). For simplicity's sake you can keep `a`, `b`, and `c` in registers `s0`, `s1`, and `s2` (the `t` registers are not safe since we are calling `printf`) and *do not* need to implement them on the stack. You may also assume that the format string has been declared in the static memory section and is labeled `fmt`. [12 points]

Student Name: _____

CS 341

④ **The datapath and other things.**

[25 points]

a) As far as we know it (and we didn't talk about it a whole lot), the MIPS datapath consists of a few important parts. One of the more essential parts is the ALU. Please describe how the MIPS processor uses the ALU when executing an `beq` instruction. [5 points]

b) MIPS is a load/store architecture. This means that only particular instructions are used to access the memory. The design is common on RISC processors. CISC processors however often allow memory to be addressed directly from say for example arithmetic instructions. What does RISC and CISC stand for? What is one benefit, and one drawback of the load/store architecture (i.e., RISC) compared to the CISC style CPU's? [5 points]

c) Using the MIPS Instruction Encoding sheet attached to this exam, convert the following MIPS assembly instruction into MIPS machine code (in hex). Show your work. [5 points]

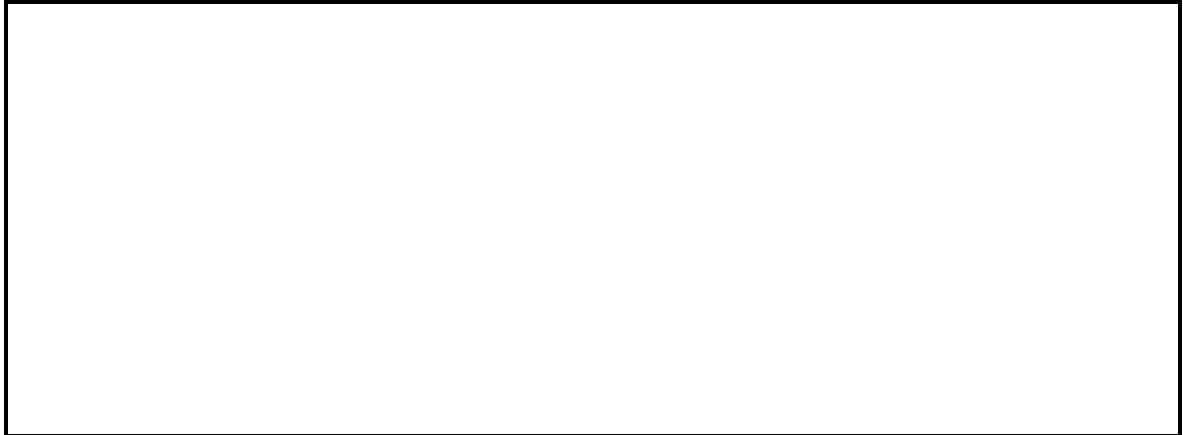
`lh $t7, -30($s5)`

Student Name: _____

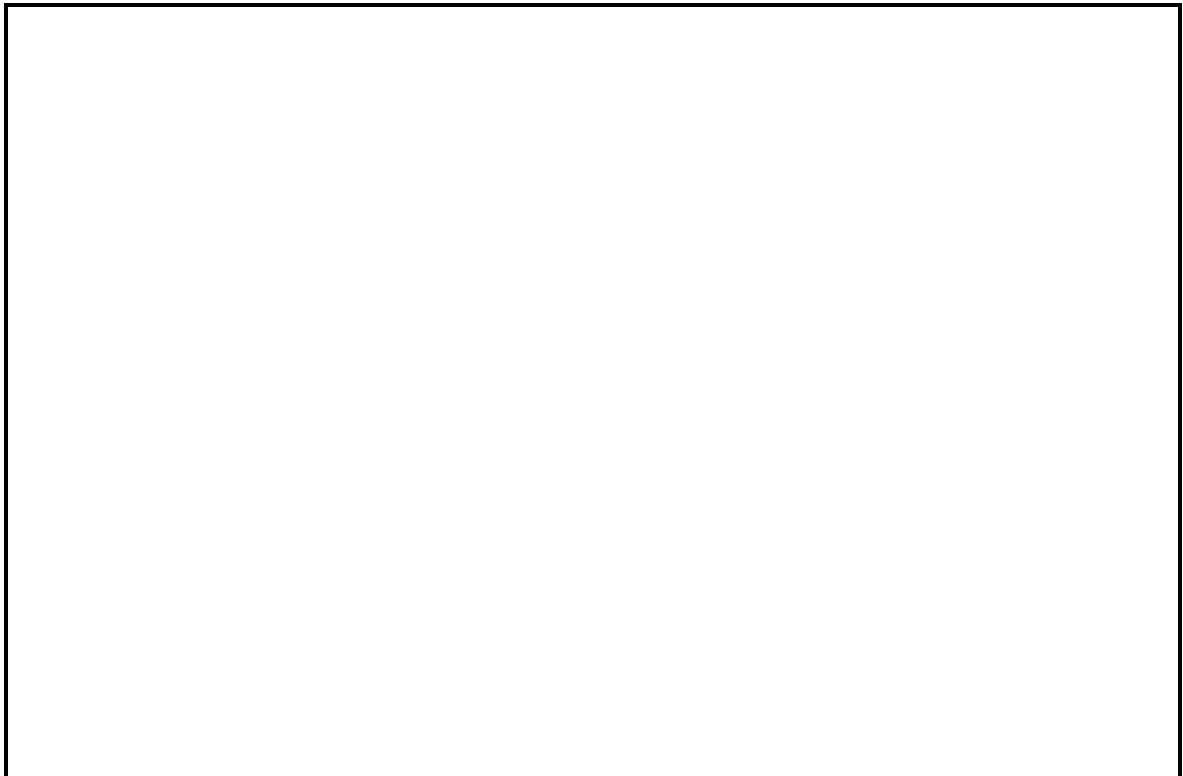
CS 341

- d)** Using the MIPS Instruction Encoding sheet attached to this exam, convert the following MIPS machine code instruction into MIPS assembly code. Registers should be given in their symbolic form, e.g. `$t0` rather than `$8`. Show your work. [5 points]

0x15E01009



- e)** Convert the following expression $((a + 3)/4 - 5) * (8 - (b/4 + c * 4))$ into 0-address assembly code. It may help to show what the stack looks like after each instruction. [5 points]



Extra Space for Answers

MIPS Instruction Encodings and Register Map

Instructions that put a result in a destination register put it in `rd` if it is specified, otherwise `rt` is used.

dec	hex	Instruction						Operation	Description
		31:26	25:21	20:16	15:11	10:6	5:0		
0	00	000000	00000	rt	rd	imm5	000000	sll	shift left logical
0	00	000000	00000	rt	rd	imm5	000010	srl	shift right logical
0	00	000000	00000	rt	rd	imm5	000011	sra	shift right arithmetic
0	00	000000	rs	rt	rd	00000	000100	sllv	sll variable
0	00	000000	rs	rt	rd	00000	000110	srlv	srl variable
0	00	000000	rs	rt	rd	00000	000111	srav	sra variable
0	00	000000	rs	00000	00000	00000	001000	jr	jump register
0	00	000000	rs	00000	rd	00000	001001	jalr	jump register and link
0	00	000000	00000	00000	00000	00000	001100	syscall	system call
0	00	000000	imm20				001101	break	raise exception
0	00	000000	00000	00000	rd	00000	010000	mfhi	move from hi
0	00	000000	rs	00000	00000	00000	010001	mthi	move to hi
0	00	000000	00000	00000	rd	00000	010010	mflo	move from lo
0	00	000000	rs	00000	00000	00000	010011	mtlo	move to lo
0	00	000000	rs	rt	00000	00000	011000	mult	multiply
0	00	000000	rs	rt	00000	00000	011001	multu	multiply unsigned
0	00	000000	rs	rt	00000	00000	011010	div	divide
0	00	000000	rs	rt	00000	00000	011011	divu	divide unsigned
0	00	000000	rs	rt	rd	00000	100000	add	add
0	00	000000	rs	rt	rd	00000	100001	addu	add unsigned
0	00	000000	rs	rt	rd	00000	100010	sub	subtract $rs-rt$
0	00	000000	rs	rt	rd	00000	100011	subu	subtract unsigned
0	00	000000	rs	rt	rd	00000	100100	and	logical and
0	00	000000	rs	rt	rd	00000	100101	or	logical or
0	00	000000	rs	rt	rd	00000	100110	xor	logical xor
0	00	000000	rs	rt	rd	00000	100111	nor	logical nor
0	00	000000	rs	rt	rd	00000	101010	slt	set less than
0	00	000000	rs	rt	rd	00000	101011	sltu	set less than unsigned
1	01	000001	rs	00000	imm16			bltz	branch on $rs < 0$
1	01	000001	rs	00001	imm16			bgez	branch on $rs \geq 0$
1	01	000001	rs	10000	imm16			bltzal	branch on $rs < 0$ and link
1	01	000001	rs	10001	imm16			bgezal	branch on $rs \geq 0$ and link
2	02	000010	imm26					j	jump
3	03	000011	imm26					jal	jump and link

dec	hex	Instruction						Operation	Description
		31:26	25:21	20:16	15:11	10:6	5:0		
4	04	000100	rs	rt	imm16			beq	branch on $rs = rt$
5	05	000101	rs	rt	imm16			bne	branch on $rs \neq rt$
6	06	000110	rs	00000	imm16			blez	branch on $rs \leq 0$
7	07	000111	rs	00000	imm16			bgtz	branch on $rs > 0$
8	08	001000	rs	rt	imm16			addi	add immediate
9	09	001001	rs	rt	imm16			addiu	add immediate unsigned
10	0A	001010	rs	rt	imm16			slti	set less than immediate
11	0B	001011	rs	rt	imm16			sltiu	slt immediate unsigned
12	0C	001100	rs	rt	imm16			andi	and immediate
13	0D	001101	rs	rt	imm16			ori	or immediate
14	0E	001110	rs	rt	imm16			xori	xor immediate
15	0F	001111	rs	rt	imm16			lui	load upper immediate
32	20	100000	rs	rt	imm16			lb	load byte
33	21	100001	rs	rt	imm16			lh	load half-word
34	22	100010	rs	rt	imm16			lwl	load word left
35	23	100011	rs	rt	imm16			lw	load word
36	24	100100	rs	rt	imm16			lbu	lb unsigned
37	25	100101	rs	rt	imm16			lhu	lh unsigned
38	26	100110	rs	rt	imm16			lwr	load word right
40	28	101000	rs	rt	imm16			sb	store byte
41	29	101001	rs	rt	imm16			sh	store half-word
42	2A	101010	rs	rt	imm16			swl	store word left
43	2B	101011	rs	rt	imm16			sw	store word
46	2E	101110	rs	rt	imm16			swr	store word right

Name	Number	Usage
zero	0	Constant 0
at	\$1	Reserved for assembler
v0-v1	\$2-\$3	Expression evaluation and results of a function
a0-a3	\$4-\$7	Argument 1-4
t0-t7	\$8-\$15	Temporary (not preserved across call)
s0-s7	\$16-\$23	Saved temporary (preserved across call)
t8-t9	\$24-\$25	Temporary (not preserved across call)
k0-k1	\$26-\$27	Reserved for OS kernel
gp	\$28	Pointer to global area
sp	\$29	Stack pointer
fp	\$30	Frame pointer
ra	\$31	Return address (used by function call)

n	Binary (2^n)	Octal (8^n)	Decimal (10^n)	Hex (16^n)
0	1	1	1	1
1	2	8	10	16
2	4	64	100	256
3	8	512	1000	4096
4	16	4096	10000	65536
5	32	32768	100000	1048576
6	64	262144	1000000	16777216
7	128	2097152	10000000	268435456
8	256	16777216	100000000	4294967296
9	512	134217728	1000000000	68719476736
10	1024	1073741824	10000000000	1099511627776
11	2048	8589934592	100000000000	17592186044416
12	4096	68719476736	1000000000000	281474976710656
13	8192	549755813888	10000000000000	4503599627370496
14	16384	4398046511104	100000000000000	72057594037927936
15	32768	35184372088832	1000000000000000	1152921504606846976
16	65536	281474976710656	10000000000000000	18446744073709551616
17	131072	2251799813685248	100000000000000000	295147905179352825856
18	262144	18014398509481984	1000000000000000000	4722366482869645213696
19	524288	144115188075855872	10000000000000000000	75557863725914323419136
20	1048576	1152921504606846976	100000000000000000000	1208925819614629174706176
21	2097152	9223372036854775808	1000000000000000000000	19342813113834066795298816
22	4194304	73786976294838206464	10000000000000000000000	309485009821345068724781056
23	8388608	590295810358705651712	100000000000000000000000	4951760157141521099596496896
24	16777216	4722366482869645213696	1000000000000000000000000	79228162514264337593543950336
25	33554432	37778931862957161709568	10000000000000000000000000	1267650600228229401496703205376
26	67108864	302231454903657293676544	100000000000000000000000000	20282409603651670423947251286016
27	134217728	2417851639229258349412352	1000000000000000000000000000	324518553658426726783156020576256
28	268435456	19342813113834066795298816	10000000000000000000000000000	5192296858534827628530496329220096
29	536870912	154742504910672534362390528	100000000000000000000000000000	83076749736557242056487941267521536
30	1073741824	1237940039285380274899124224	1000000000000000000000000000000	1329227995784915872903807060280344576
31	2147483648	9903520314283042199192993792	10000000000000000000000000000000	21267647932558653966460912964485513216

Table 1: A power table!