McGill

August 2017
Supplemental/Deferred Examination

Introduction to Computer Systems

COMP-273

August 2017

Examiner: Joseph Vybihal Assoc Examiner:

Student Name:		McGill ID:									

INSTRUCTIONS:

- This is a **CLOSED BOOK** examination.
- You are permitted **TRANSLATION** dictionaries ONLY.
- **STANDARD CALCULATOR** permitted ONLY.
- This examination is **PRINTED ON ONE SIDE** of the paper
- This examination paper **MUST BE RETURNED**
- You are permitted to write your answers in either **English or French**
- Write your answers on the **exam paper** providedd
- Attemp all questions, **part marks** will be assigned, show your work.

Grading

Section	Grade	Your Mark
Question 1: Definitions & Problems	20	
Question 2: Assembler Programming	30	
Question 3: Circuit Drawing	30	
Question 4: Calculations	20	
Total	100 %	

Question 1: Definitions

Answer the following questions in the exam booklet provided. (Crib sheets in the back pages)

1. The MIPS CPU has two co-processors. (1) Name the two co-processors. (2) Provide example instructions for how to use these co-processors (provide an instruction that does one of the following: send/get data, use it/compute with it).

2. In a 4-way 4-word block TLB (translation lookaside buffer) of an interesting but small size of your choice where the TLB is large enough to store an entire array: (1) describe how an array of 100 integers would be addressed and stored if there was no competition for the cache space. (2) Describe how competing resources would affect the state of this array (give one example using a competing array of 100 characters).

3. Provide answers for the following:
   (A) Convert this assembler instruction into machine code: `BEQ $t1, $t2, EXIT`
   (B) Convert this assembler instruction into machine code: J NEXT

4. Assume in Parallel C we have the commands PARBEGIN and PAREND and PARSHARE to describe, in a programming language, how to execute code in parallel on a multi-core CPU. PARBEGIN and PAREND are like open and close curly brackets defining a block of code that executes in parallel. Every instruction, line of code, in that segment runs in parallel. PARESHARE defines data that all parallel executing instructions can access at the same time. Assume we have the following code:

   ```
   PARBEGIN
     PARSHARE int array[] = {10,12,1,17, … assume 1000 numbers };
     boolean x = find(key, 0, 500);     // search array from cell 0 to 500 for key
     boolean y = find(key, 501, 999); // search array from cell 501 to 999 for key
   PAREND
   ```

   Describe how a multi-core CPU would execute this program. (A) Describe cache loading and data sharing. (B) Describe how the code would execute. (C) Describe how the OS passing control to the cores. Assume a 4 core CPU that does not support hyper threading. This is a standard MIPS type CPU.

Question 2: Assembler Programming

Write the following assembler function and provide an example call. Remember a function follows the C calling convention described in class and in your assignment. Assume this function is called by a main program with the following C signature: x = max(a,b,c); Your function must uses the proper calling, local memory, and returning conventions. Do the following: (1) in MIPS write a code snip that shows an example of how the function would be called from the main program. You do not need to define the .data area. (2) Then write in MIPS the complete function as per conventions.

Question 3: Circuit Drawing

You can only use the following tools: and-gate, or-gate, not-gate, wires, and a flip-flop black box.

Construct the following circuit: create four 2-bit registers (labelled A, B, C, and D) and three 1-bit registers (labelled S, T and U). These registers are connected to a single uni-directional common bus made from 2 wires. Registers A, B, C and D send and receive data to/from the bus. Register S selects which source (A or B) will send data to which destination (C or D) registers. If S is 0 then A sends to C, otherwise B sends to D. Register T does the opposite. When T is 0 then C sends to A, otherwise D sends to B. The register U determines which register, S or T, the circuit executes. If U is 0 then S is used, otherwise T is used.

Question 4: Calculations

1. Count the following (if you need to make assumptions please state them):
   (A) Count the minimum number of move operations (load and store operations) to copy 1000 integer numbers from one array in RAM to another array in RAM from the point-of-view of the CPU.
   (B) Assume a program is using polling to copy 1000 integer from the keyboard to an array in RAM. From the point-of-view of the CPU, how many move operations are required. In this case, provide a formula that provides the count.
   (C) Assume a program is reading 1000 integer numbers from the hard drive into an array in RAM. The hard drive uses interrupts to load data into its private 1000-byte buffer. Provide a formula expressing a count for the number of move operations required from the point-of-view of the CPU to get this data from the hard drive into the array in RAM.
   (D) Redo (C) but assume the computer has a DMI between the hard drive and RAM. Provide a formula that expresses a count of move operations from the point-of-view of the CPU to get the 1000 integer from the hard drive into the array in RAM.

2. Assume you have a software program that runs science experiments with the following requirements: computes equations 50% of the time, waits for input from an external device 20% of the time, save data to RAM 20% of the time, saves data to the hard drive 10% of the time. The computer you are currently using as the following specifications: 1 GHz CPU, 4 MB of RAM, 500 GB hard drive. You have only enough money to buy one piece of new replacement hardware. What should you buy from the following: a 2 GHz CPU, new RAM that runs 20% faster, a new hard drive that runs twice as fast as what you have now. Show your work. You must compute your solution. State any assumptions you needed to make.

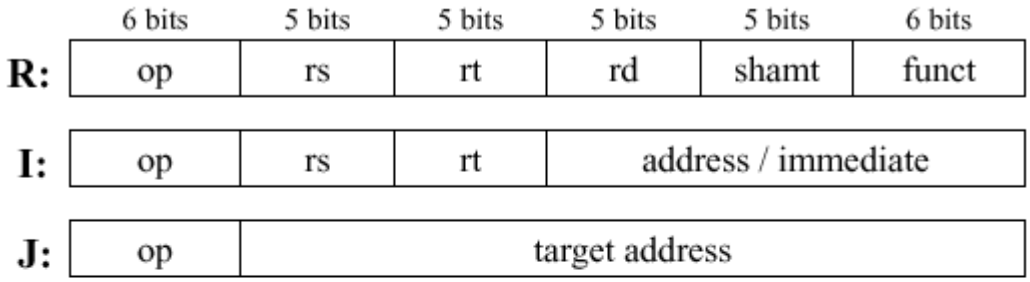
3. If an interrupt occurs every half second on a 500 MHz CPU and takes 1000 cs to process, how long would it take to execute a program that needs to run for 1,000,000,000 cs? Assume cs is equivalent to 2 instructions. State any other correct assumptions you have made to answer this question.

Bonus Question

First bonus question: Name the principal female character in the story?
Second bonus question: What was the name of the computer they were building?

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add $1,$2,$3	$1 = $2 + $3	3 operands; exception possible
	subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands; exception possible
	add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant; exception possible
	add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands; no exceptions
	subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands; no exceptions
	add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant; no exceptions
	Move fr. copr. reg.	mfc0 $1,$epc	$1 = $epc	Used to get exception PC
	multiply	mult $2,$3	Hi, Lo = $2 ¥ $3	64-bit signed product in Hi, Lo
	multiply unsigned	multu $2,$3	Hi, Lo = $2 ¥ $3	64-bit unsigned product in Hi, Lo
	divide	div $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Lo = quotient, Hi = remainder
	divide unsigned	divu $2,$3	Lo = $2 ÷ $3, Hi = $2 mod $3	Unsigned quotient and remainder
	Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
	Move from Lo	mflo $1	$1 = Lo	Use to get copy of Lo
Logical	and	and $1,$2,$3	$1 = $2 & $3	3 register operands; logical AND
	or	or $1,$2,$3	$1 = $2 | $3	3 register operands; logical OR
	and immediate	andi $1,$2,100	$1 = $2 & 100	Logical AND register, constant
	or immediate	ori $1,$2,100	$1 = $2 | 100	Logical OR register, constant
	shift left logical	sll $1,$2,10	$1 = $2 << 10	Shift left by constant
	shift right logical	srl $1,$2,10	$1 = $2 >> 10	Shift right by constant
Data transfer	load word	lw $1,100($2)	$1 = Memory[$2+100]	Data from memory to register
	store word	sw $1,100($2)	Memory[$2+100] = $1	Data from register to memory
	load upper imm.	lui $1,100	$1 = 100 x 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100	Equal test; PC relative branch
	branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100	Not equal test; PC relative
	set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0	Compare less than; 2's complement
	set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0	Compare < constant; 2's comp.
	set less than uns.	sltu $1,$2,$3	if ($2 < $3) $1=1; else $1=0	Compare less than; natural number
	set l.t. imm. uns.	sltiu $1,$2,100	if ($2 < 100) $1=1; else $1=0	Compare < constant; natural
Unconditional jump	jump	j 10000	go to 10000	Jump to target address
	jump register	jr $31	go to $31	For switch, procedure return
	jump and link	jal 10000	$31 = PC + 4; go to 10000	For procedure call



op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode (function code)
address: offset for load/store instructions (+/-2^{15})
immediate: constants for immediate instructions

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		

FIGURE A.17 System services.

MIPS floating-point operands

Name	Example	Comments
32 floating-point registers	`$f0, $f1, $f2, . . . , $f31`	MIPS floating-point registers are used in pairs for double precision numbers.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential words differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS floating-point assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	`add.s $f2,$f4,$f6`	$f2 = $f4 + $f6	FP add (single precision)
	FP subtract single	`sub.s $f2,$f4,$f6`	$f2 = $f4 – $f6	FP sub (single precision)
	FP multiply single	`mul.s $f2,$f4,$f6`	$f2 = $f4 × $f6	FP. multiply (single precision)
	FP divide single	`div.s $f2,$f4,$f6`	$f2 = $f4 / $f6	FP divide (single precision)
	FP add double	`add.d $f2,$f4,$f6`	$f2 = $f4 + $f6	FP add (double precision)
	FP subtract double	`sub.d $f2,$f4,$f6`	$f2 = $f4 – $f6	FP sub (double precision)
	FP multiply double	`mul.d $f2,$f4,$f6`	$f2 = $f4 × $f6	FP multiply (double precision)
	FP divide double	`div.d $f2,$f4,$f6`	$f2 = $f4 / $f6	FP divide (double precision)
Data transfer	load word copr. 1	`lwc1 $f1,100($s2)`	$f1 = Memory[$s2 + 100]	32-bit data to FP register
	store word copr. 1	`swc1 $f1,100($s2)`	Memory[$s2 + 100] = $f1	32-bit data to memory
Conditional branch	branch on FP true	`bc1t 25`	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
	branch on FP false	`bc1f 25`	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
	FP compare single (eq,ne,lt,le,gt,ge)	`c.lt.s $f2,$f4`	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than single precision
	FP compare double (eq,ne,lt,le,gt,ge)	`c.lt.d $f2,$f4`	if ($f2 < $f4) cond = 1; else cond = 0	FP compare less than double precision

Remaining MIPS I	Name	Format	Pseudo MIPS	Name	Format
exclusive or ($rs \oplus rt$)	xor	R	move	move	rd,rs
exclusive or immediate	xori	I	absolute value	abs	rd,rs
nor ($\neg(rs \vee rt)$)	nor	R	not ($\neg rs$)	not	rd,rs
shift right arithmetic	sra	R	negate *(signed or unsigned)*	neg*s*	rd,rs
shift left logical variable	sllv	R	rotate left	rol	rd,rs,rt
shift right logical variable	srlv	R	rotate right	ror	rd,rs,rt
shift right arith. variable	srav	R	mult. & don't check oflw *(signed or uns.)*	mul*s*	rd,rs,rt
			multiply & check oflw *(signed or uns.)*	mulo*s*	rd,rs,rt
move to Hi	mthi	R	divide and check overflow	div	rd,rs,rt
move to Lo	mtlo	R	divide and don't check overflow	divu	rd,rs,rt
load halfword	lh	I	remainder *(signed or unsigned)*	rem*s*	rd,rs,rt
load halfword unsigned	lhu	I	load immediate	li	rd,imm
store halfword	sh	I	load address	la	rd,addr
load word left *(unaligned)*	lwl	I	load double	ld	rd,addr
load word right *(unaligned)*	lwr	I	store double	sd	rd,addr
store word left *(unaligned)*	swl	I	unaligned load word	ulw	rd,addr
store word right *(unaligned)*	swr	I	unaligned store word	usw	rd,addr
branch on less than zero	bltz	I	unaligned load halfword *(signed or uns.)*	ulh*s*	rd,addr
branch on less or equal zero	blez	I	unaligned store halfword	ush	rd,addr
branch on greater than zero	bgtz	I	branch	b	Label
branch on ≥ zero	bgez	I	branch on equal zero	beqz	rs,L
branch on ≥ zero and link	bgezal	I	branch on ≥ *(signed or unsigned)*	bge*s*	rs,rt,L
branch on < zero and link	bltzal	I	branch on > *(signed or unsigned)*	bgt*s*	rs,rt,L
jump and link register	jalr	R	branch on ≤ *(signed or unsigned)*	ble*s*	rs,rt,L
return from exception	rfe	R	branch on < *(signed or unsigned)*	blt*s*	rs,rt,L
system call	syscall	R	set equal	seq	rd,rs,rt
break *(cause exception)*	break	R	set not equal	sne	rd,rs,rt
move from FP to integer	mfc1	R	set greater or equal *(signed or unsigned)*	sge*s*	rd,rs,rt
move to FP from integer	mtc1	R	set greater than *(signed or unsigned)*	sgt*s*	rd,rs,rt
FP move *(s or d)*	mov.*f*	R	set less or equal *(signed or unsigned)*	sle*s*	rd,rs,rt
FP absolute value *(s or d)*	abs.*f*	R	set less than *(signed or unsigned)*	sle*s*	rd,rs,rt
FP negate *(s or d)*	neg.*f*	R	load to floating point *(s or d)*	l.*f*	rd,addr
FP convert *(w, s, or d)*	cvt.*ff*	R	store from floating point *(s or d)*	s.*f*	rd,addr
FP compare un *(s or d)*	c.xn.*f*	R			

Name	Register Number	Usage	Preserved on call
$zero	0	the constant value 0	n.a.
$at	1	reserved for the assembler	n.a.
$v0-$v1	2-3	value for results and expressions	no
$a0-$a3	4-7	arguments (procedures/functions)	yes
$t0-$t7	8-15	temporaries	no
$s0-$s7	16-23	saved	yes
$t8-$t9	24-25	more temporaries	no
$k0-$k1	26-27	reserved for the operating system	n.a.
$gp	28	global pointer	yes
$sp	29	stack pointer	yes
$fp	30	frame pointer	yes
$ra	31	return address	yes