COMP 273
Introduction to Computer Systems

COMP 273

Final Exam Review

Prof. Joseph Vybihal

Announcements

- Final Exam:
  - Monday, Dec 12, 2 PM
  - Sample final posted
- Final exam office hours
  - Open door & by appointment

Topics on Exam

- MIPS Programming
- Peripheral Interfacing
- Programming Interrupts
- Cache circuit architecture
- Performance & other mathematical calcs.
- Circuit knowledge
  - Floating point programming & circuits
  - Basic circuit architecture

Concepts on Exam

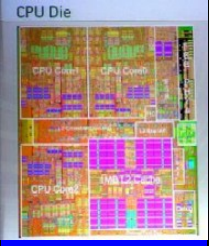
- Sys-calls
- Run-time stack
- Saving registers with subroutines
  - C convention, MIPS convention
- Dynamic Memory
- Basic circuit diagram reading
- Hardware interrupts
- Peripherals
- Performance

Final Exam Format

- 4 questions
  - Definitions (4 of them) – ¾ term
  - Two MIPS programming questions
  - Calculations (3 of them) – all term
  - (2 bonus questions)
- Rules
  - Closed book, no calculators
  - Answer on exam
  - Part marks given
  - Teacher supplied help sheets

What is covered?

- Everything after the midterm
  - MIPS assembly language programming
  - Circuit interpretation
  - MIPS features
    - caches, virtual memory, dynamic memory, recursion / stacks
    - interrupts and exception handling
    - memory mapped I/O
    - Buses, synchronous vs. asynchronous I/O
- Things you should know but are not tested
  - Digital Math
  - Digital data formats
- Things to know, calculations:
  - Amdahl's Law
  - Polling & Interrupt overhead
  - Cache Performance

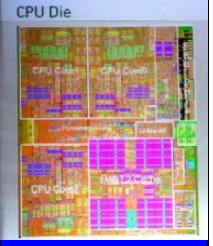


COMP 273
Introduction to Computer Systems

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

Given on Exam



MIPS floating-point operands

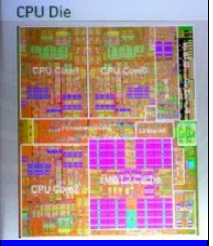
Name	Example	Comments
32 floating-point registers	`$f0, $f1, $f2, . . . , $f31`	MIPS floating-point registers are used in pairs for double precision numbers.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so 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
Condi-tional 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

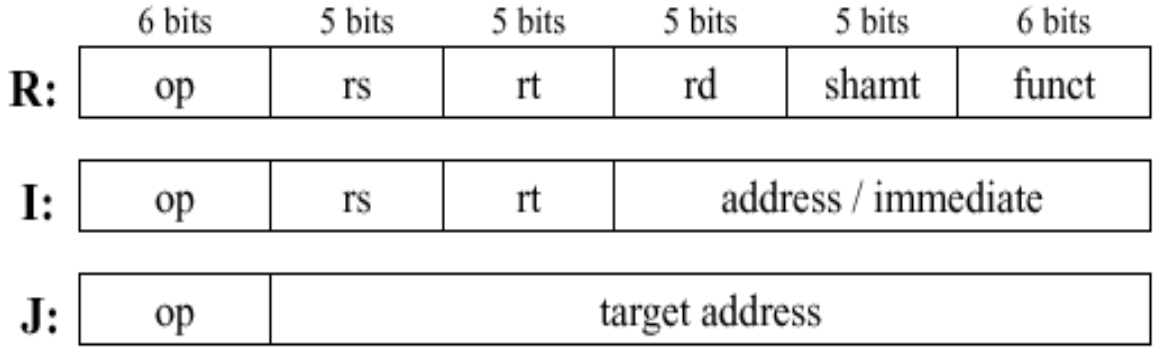
Remaining MIPS I	Name	Format
exclusive or ($rs \oplus rt$)	xor	R
exclusive or immediate	xori	I
nor ($\neg(rs \vee rt)$)	nor	R
shift right arithmetic	sra	R
shift left logical variable	sllv	R
shift right logical variable	srlv	R
shift right arith. variable	srav	R
move to Hi	mthi	R
move to Lo	mtlo	R
load halfword	lh	I
load halfword unsigned	lhu	I
store halfword	sh	I
load word left *(unaligned)*	lwl	I
load word right *(unaligned)*	lwr	I
store word left *(unaligned)*	swl	I
store word right *(unaligned)*	swr	I
branch on less than zero	bltz	I
branch on less or equal zero	blez	I
branch on greater than zero	bgtz	I
branch on ≥ zero	bgez	I
branch on ≥ zero and link	bgezal	I
branch on < zero and link	bltzal	I
jump and link register	jalr	R
return from exception	rfe	R
system call	syscall	R
break *(cause exception)*	break	R
move from FP to integer	mfc1	R
move to FP from integer	mtc1	R
FP move *(s or d)*	mov*f*	R
FP absolute value *(s or d)*	abs*f*	R
FP negate *(s or d)*	neg*f*	R
FP convert *(w, s, or d)*	cvt*ff*	R
FP compare un *(s or d)*	c.xn*f*	R

Pseudo MIPS	Name	Format
move	move	rd,rs
absolute value	abs	rd,rs
not ($\neg rs$)	not	rd,rs
negate *(signed or unsigned)*	neg*s*	rd,rs
rotate left	rol	rd,rs,rt
rotate right	ror	rd,rs,rt
mult. & don't check oflw *(signed or uns.)*	mul*s*	rd,rs,rt
multiply & check oflw *(signed or uns.)*	mulo*s*	rd,rs,rt
divide and check overflow	div	rd,rs,rt
divide and don't check overflow	divu	rd,rs,rt
remainder *(signed or unsigned)*	rem*s*	rd,rs,rt
load immediate	li	rd,imm
load address	la	rd,addr
load double	ld	rd,addr
store double	sd	rd,addr
unaligned load word	ulw	rd,addr
unaligned store word	usw	rd,addr
unaligned load halfword *(signed or uns.)*	ulh*s*	rd,addr
unaligned store halfword	ush	rd,addr
branch	b	Label
branch on equal zero	beqz	rs,L
branch on ≥ *(signed or unsigned)*	bge*s*	rs,rt,L
branch on > *(signed or unsigned)*	bgt*s*	rs,rt,L
branch on ≤ *(signed or unsigned)*	ble*s*	rs,rt,L
branch on < *(signed or unsigned)*	blt*s*	rs,rt,L
set equal	seq	rd,rs,rt
set not equal	sne	rd,rs,rt
set greater or equal *(signed or unsigned)*	sge*s*	rd,rs,rt
set greater than *(signed or unsigned)*	sgt*s*	rd,rs,rt
set less or equal *(signed or unsigned)*	sle*s*	rd,rs,rt
set less than *(signed or unsigned)*	sle*s*	rd,rs,rt
load to floating point *(s or d)*	l*f*	rd,addr
store from floating point *(s or d)*	s*f*	rd,addr



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.



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

when op is 0 or 1 the operation is determined by other fields

op

000000	
000001	
000010	j
000011	jal
000100	beq
000101	bne
000110	blez
000111	bgtz
001000	addi
001001	addiu
001010	slti
001011	sltiu
001100	andi
001101	ori
001110	xori
001111	lui
100000	lb
100001	lh
100010	lwl
100011	lw
100100	lbu
100101	lhu
100110	lwr
100111	
101000	sb
101001	sh
101010	swl
101011	sw

rt

00000	bltz
00001	bgez
.	
.	
.	
10000	bltzal
10001	bgezal

funct

000000	sll
000001	
000010	srl
000011	sra
000100	sllv
000101	
000110	srlv
000111	srav
001000	jr
001001	jalr
001010	
001011	
001100	syscall
001101	break
001110	
001111	
010000	mfhi
010001	mthi
010010	mflo
010011	mtlo
011000	mult
011001	multu
011010	div
011011	divu
100000	add
100001	addu
100010	sub
100011	subu
100100	and
100101	or
100110	xor
100111	nor
101000	
101001	
101010	slt
101011	sltu

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

COMP 273
Introduction to Computer Systems

Questions?

Discussion Problems

- Assembler:
  - O(n) malloc & free implementation
    - Is there an O(1)?
  - Printer polling with address and status bit
  - Direct output to video screen buffer
    - OS buffered
    - Program direct

Problems

- How long would it take to write a 100 character string to a printer – command buffer set to 0 for write:
  - Using direct interface to a 200 byte on-board printer buffer? (a move is 10cs)
  - Using polling and no buffer? (polling overhead of 30cs)
  - Using interrupts and no buffer? (overhead set 30cs, response 30cs) (does this work?)
  - Using DMA and interrupts with buffer?

Research/Project Opportunities

- Joseph Vybihal
  - The Prometheus Project
    - AI thinking simulation
    - Mobile robotics
    - Vision

COMP 400
COMP 396
Volunteering
Summer Projects
Work Study