CS202 Computer Organization

Spring 2020

Midterm Examination

Date: April 28, 2020 **Time: 16:00 – 18:00**

Student ID: **Name:_____**

1. (30 points) Simple Choice. Please select one best answer from A B C and D.

   1) In the following items, _____ can reduce the program execution time.

      i) Increase the clock rate ii) improve the datapath iii）optimize the compiler

      A. only i and ii B. only i and iii C. only ii and iii D. i, ii and iii

   2) Computer M1 and M2 has the same ISA, their clock rate is 1.2GHz and 1.5GHz respectively. The CPI of running a program P on M1 and M2 is 1 and 2 respectively, then, the ratio of the execution time of running P on M1 and M2 is _____.

      A. 0.4 B. 0.625 C. 1.6 D. 2.5

   3) Assuming that there are four integers, each of them is in an 8-bit register in format of 2's complement. r1=0xFE, r2=0xF2, r3=0xF8, r4=0x90. If the result is also in an 8-bit register, the operation of _____ will generate overflow.

      A. r1 x r2 B. r2 x r3 C. r1 x r4 D. r2 x r4

   4) x and y are two float numbers stored in 32-bit registers f1 and f2 respectively, (f1) = 0x B0C00111, (f2)=0xCC900110, then x and y follows_____.

      A. x<y, sign(x)=sign(y) B. x<y, sign(x)≠sign(y)

      C. x>y, sign(x)=sign(y) D. x>y, sign(x) ≠sign(y)

   5) Assume that in a pipeline CPU, each instruction can be divided into three stages: IF ID EXE, each stage cost Δt execution time. If the CPU employ 4-multiple issue pipeline, assume that 24 instructions are consecutively executed without any stalls, then the total execution time for them is_____.

      A. 3Δt B. 5Δt C. 7Δt D. 8Δt

   6) For a 4-stage pipeline CPU, with a 1.03GHz clock rate, each stage cost one clock cycle. 100 instructions are consecutively executed in the pipeline without any stalls, calculate the throughput of this CPU for running the above 100 instructions.

      A. $0.25*10^9$ instructions/second B. $0.97*10^9$ instructions/second

      C. $1.0*10^9$ instructions/second D. $1.03*10^9$ instructions/second

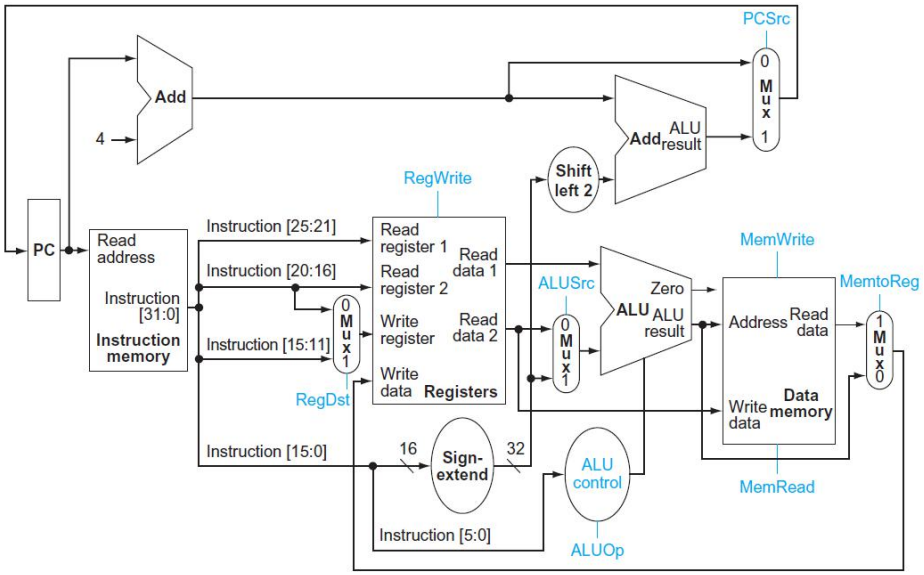
   7) What can be stored in the register module of CPU?

      A. Only data and address can be stored in it.

B. Only data can be stored in it, address can not be stored in it.

C. Neither data nor address can be stored in it.

D. Data, address and instructions can all be stored in it.



8) Which of the following statement is not correct?

A. RegWrite is 1 in lw B. ALUSrc is 1 in addi

C. MemtoReg is 1 in add D. MemWrite is 1 in sw

9) Sign extension is not needed in _____ instruction.

D. sll B. addi C. bne D. lw

10) To exploit instruction-level parallelism, which of the following approaches are not hardware-based approach?

A. superscalar B. very long instruction word C. dynamic multiple issue D. reorder buffer

2. (10 points). You are designing an embedded processor. Based on an analysis of the software that it will run, you find the following mix of instructions, which have the specified execution time in your current design:

Instructions	Percentage	Time
load	12%	4 cycles
store	10%	8 cycles
branch	18%	3 cycles
add	60%	1 cycle

1) (2 points) What is the CPI of your processor on this mix of instructions?

2) (2 points) If the clock rate of your processor is 40MHz, and the total number of instructions of the software is 5000. Calculate the execution time for the software to run in your processor.

3) (2 points) Based on your design analysis, you figure out that you can halve the cycle latency of any single category of instruction, although you will need to increase the cycle time by 10%. Should you make this change, and if so, what category of instruction should you speed up?

4) (2 points) What is the CPI of your new design?

5) (2 points) What is the speedup of your revised design over the original one?

3. (8 points) At the end of executing the following MIPS instruction sequence, specify the contents of the following registers

```
        addi $s0, $zero, 6
        addi $v0, $zero, -3
        add $s1, $s0, $v0
        beq $s1, $v0, Skip
        srl $s2, $s1, 1
    Skip:
        or $v0, $v0, $s2
```

Answer:

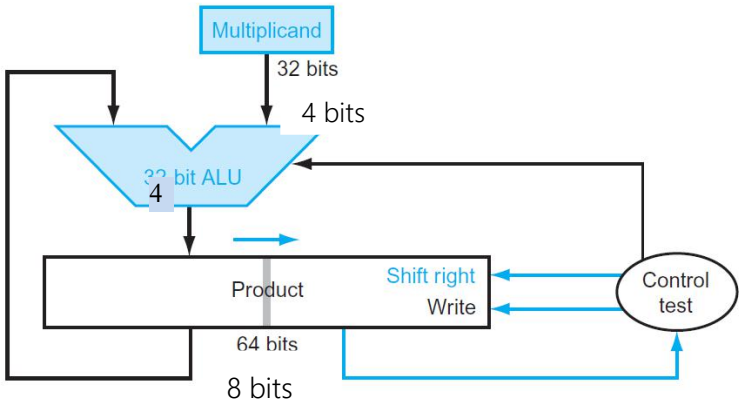
Register $s0 =

Register $v0 =

Register $s1 =

Register $s2 =

4. (a) (8 points) Calculate the product of 6 × 5 using the hardware described below, with the multiplicand equals to 6. Both multiplicand and multiplier are unsigned 4-bit integers. Please show the contents of each register on each step and the final result.

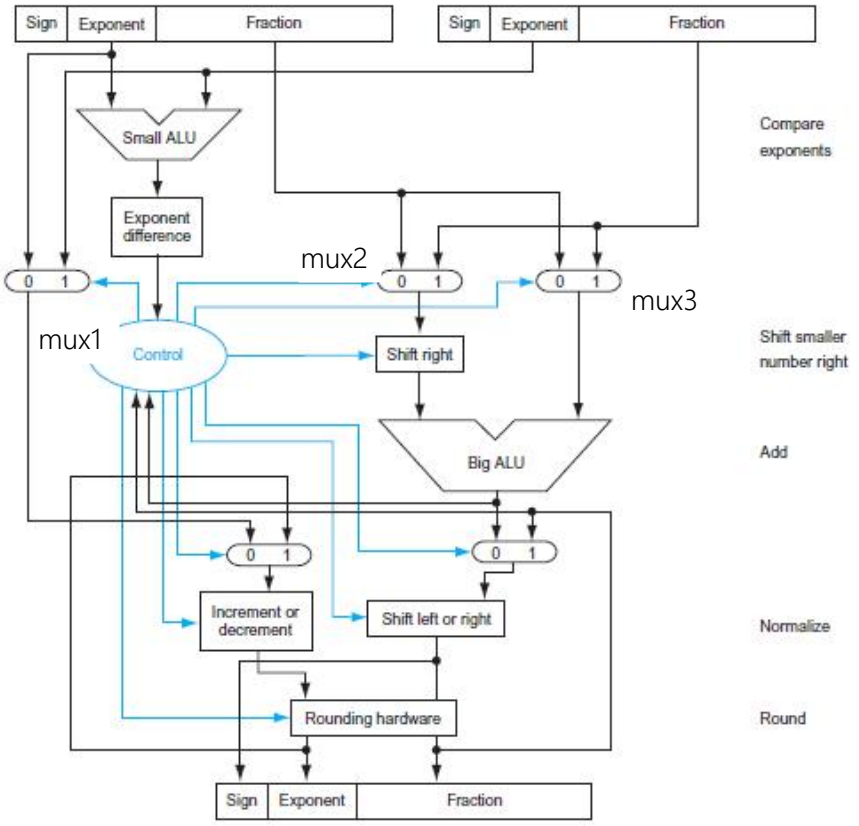


Answer:

Iteration	Multiplicand	Product
0		
1		
...		

(b) (9 points) Consider a 16-bit floating-point number format as follows, the left most bit is still the sign bit, the exponent is 5 bits wide and has a bias of 16 (the real exponent equals to the number stored in exponent part minus 16), and the fraction part is 10 bits long. A hidden 1 in the fraction part is assumed. Write down the bit pattern to represent -0.84375 in decimal. Calculate the range and relative precision of this 16-bit floating point format, assuming the all-one bit stream and all-zero bit stream in the exponent domain are reserved.

(c) (7 points) In the following adder for floating points, assume the left floating-point number is 2.625, the right floating-point number is -0.125. 1) write down the control input for mux1, mux2 and mux3. Please show the calculation procedure explaining how you get the results. 2) In the normalize stage, should the operation in the module "Increment or decrement" be increment or decrement? Should the operation in the module "shift left or right" be left or right? Why?



5. For the C code: "for (int i=0; i<N; i++) result = result + data[i]", assume that the values of result, i, and N are in registers $s0, $s1 and $s6 respectively. Also, assume that register $s3 holds the base address of the array data. The code is stored in the instruction memory started from address 0x08048100. The compiled MIPS code is as follows:

Instruction	Address	MIPS code	Addressing mode
1	0x08048100	loop: sll $s4, $s1, 2	
2	0x08048104	add $s4, $s4, $s3	
3	0x08048108	lw $s5, 0(s4)	
4	0x0804810C	add $s0, $s0, $s5	
5	0x08048110	addi $s1, $s1, 1	
6	0x08048114	bne $s1, s6, loop	

Answer the following questions, show the steps if necessary.

1) (6 points) Please give the addressing mode for each instruction.
2) (6 points) Please give the 32-bit machine code for instruction 5 "addi $s1, $s1,1" and instruction 6 "bne $s1, s6, loop".
3) (2 points) How many bits does each element of the array data[i] occupies in the memory?

4) (6 points) Assume that the CPU works in a 5-stage pipeline IF-ID-EXE-MEM-WB without forwarding, which instructions will be stalled because of data hazards? Which instructions will be stalled because of control hazards?
5) (6 points) If forwarding is implemented in the hardware, the stalls before which instructions can be eliminated? The stalls before which instructions can be reduced? Can you use instruction reordering to further reduce the stalls? How?
6) (2 points) To reduce the stalls caused by control hazards, what method can be used?

Appendix:

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

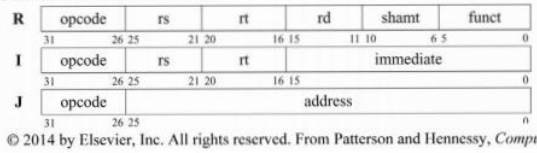
①

CORE INSTRUCTION SET

NAME, MNEMONIC		FORMAT	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

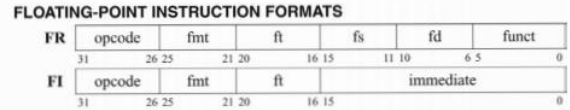


ARITHMETIC CORE INSTRUCTION SET

②

NAME, MNEMONIC		FORMAT	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