2^{20} 2^{24} $2^{16}-1$ 2^{15}

國立交通大學 資訊工程系 計算機組織 第一次考試 (2010/10/21 10:10AM~12:20AM)

請注意：1. 不可看書及任何參考資料。 2. 請詳述設計或計算過程。

99 − 3. 請依題號作答，否則高題號答案出現後，之後的低題號答案將不予計分。

李毅郎 計組

midterm

PART I:

1 **Multiple Choices (12%)** (One comment per correct answer is required. Correct answers with no comments will be discounted. Incorrect answers with reasonable comments will earn partial credits)

Luyle

1. (4%) In which of the following MIPS instructions, the ***rt*** field designates the destination register.
   - (a) Load word (lw)
   - (b) Store word (sw)
   - (c) Branch on equal (beq)
   - (d) Add with immediate (addi)

2. (4%) Which of the following objects have identical binary representation no matter the machine is little endian or big endian.

   0000 0001 0000 0001

   1111 1111 1111 1110

   - (a) 2's complement number -1 1
   - (b) int i=0xABBAABBA 1010 1011 1011 1010
   - (c) 1's complement number -1
   - (d) A C null pointer

   0000 0000 0000 0000 0100 0000 0000 0110 0000

3. (4%) Assume the current value of PC is 0x00000060, can you use the following to go to:
   - (a) a single branch instruction to get to the address 0x00040060?
   - (b) a single branch instruction to get to the address 0xFFFFFF00?
   - (c) a single jump instruction to get to 0x00040060?
   - (d) a single jump instruction to get to 0xFFFFFF00? $\div 2^{-1}$ $\overline{2^{-1}} \times 2$

Yes/No questions (12%) with one short comment. No credits will be given without any explanation.

1) (2%) The power consumption of a computer is not close to zero even when its CPU loading approaches zero.
2) (2%) CPUs with powerful instruction set mean higher performance.
3) (2%) The division of any binary number by 2^n can be performed with a right shift by n bits.
4) (2%) Instructions addi, lb, and beq all need to perform sign extension.
5) (2%) The frame pointer is more suited to locate an existing value in stack than stack pointer.
6) (2%) Intel has announced that the project of designing new CPUs operating on higher clock rate has been postponed, which implies semiconductor technology can not shrink devices any more.

210

PART II: Question sets

124 124/56 $\frac{CPI}{PCI}$ × Ins = $\frac{CPU}{Cycle}$ $\frac{Cycle}{CPI}$

1. (6%)

Use the definition of Millions Instructions Per Second (MIPS) to derive suited formula to show why it is not proper to use MIPS as a performance metric. **(You have to derive a formula (3%) and use this formula to explain your reason (3%))**

To & 4   11 0 / 10 0 xor / 01 0 / 00 0

PCI CPI

2. (19%)

The table below shows the number of instructions of two applications compiled by two compilers for a program on three different machines. Machine A has a clock rate of 4 GHz, machine B, 8 GHz, and machine C, 2 GHz. Three machines have the same three instruction types and the required number of cycle for each instruction type is: Machine A: FP 4, Int 2 and L/S 2, Machine B: FP 6, int 1 and L/S 1, and Machine C: FP 5, int 2 and L/S 1. (*must show computation to get full credit*)

		Inst. no		
		FP	Int	L/S
Compiler 1	Ap1 by C1	4.0E+9 (4×10^9)	3.0E+10 (3×10^{10})	2.0E+9 (2×10^9)
	Ap2 by C1	4.0E+10	8.0E+9	4.0E+9
Compiler 2	Ap1 by C2	2.0E+10	8.0E+9	8.0E+9
	Ap 2 by C2	1.2E+10	1.6E+10	8.0E+10

A : 4GHz
B . 8GHz
C : 2GHz

(a) (4%) If the workload is to run both applications once a week, please list workload times using two compilers on machine A.

(b) (6%) Consider the applications compiled by compiler 2. If application 1 must run four times as often as application 2 in a week, please list the workload runtime of each machine?

(c) (3%) Consider the application 2 compiled by compiler 2. Please list the average CPI of machine A.

(d) (6%) Consider the workload in (ii) (b). Can we improve Machine C's L/S instruction such that Machines B and C have the same workload time? (3%) (explain why) If we refine floating point and integer operations of Machine C to perform five and two times faster than before, what's the workload time of new Machine C? (3%)

0 − 1   0 0   92   1111 + 1111 = 1110

3. **(10%)** The overflow of 8-bit addition A+B=Sum (A=$A_7 A_6 \ldots A_0$, B= $B_7 B_6 \ldots B_0$, S= $S_7 S_6 \ldots S_0$, and carry $C_8 C_7 C_6 \ldots C_0$) can be summarized as the following conditions.

A_7 0 0   0 + 1 $\frac{240}{4} = 6$

	+/-	A	B	Sum	A_7	B_7	S_7
✓ 1	A+B	+	+	—	(b)		
✓ 2	A+B	—	—	+	A_7=1, B_7=1, and S_7=0		
✓ 3	A-B	+	—	—	Same as 1		
4	A-B	—	+	+	Same as 2		

12 8 40

(a) (3%) Give an overflow example for an 8-bit addition in case 2.

(b) (3%) In fact, the 2nd case gives C_8=1 and C_7=0, because A_7=1, B_7=1, and S_7=0. Express the other three cases in terms of A_7, B_7, S_7, C_8, and C_7.

(c) (4%) Show the overall overflow condition in terms of C_8 and C_7.

overflow = $C_7 \oplus C_8$

ex: ~~[illegible]~~ [illegible] when [illegible]

$C_8 = 1$ .. $C_8 = 0$

4. **(12%)** Carry-lookahead adder
   (a) What are basic ideas of carry-lookahead adder by using *generate* and *propagate*?
   (b) Show the delays of a 4-bit carry-lookahead adder and a 4-bit ripple carry adder.
   (c) Show the gate delay of a 16-bit adder by two-level carry-lookahead scheme and explain why. (Assume any AND, OR, or XOR gate takes only one gate delay.) 4bit

AND-OR = 2 CLA ALU $\boxed{2}$

5. **(8%)** The following figure shows a C program and its MIPS assembly program.

```
unsigned int fib(unsigned int n) {
If (n < 2) return(n);
else return (fib(n-1) + fib(n-2));
}
```

```
fib:
addi $sp,$sp, -12
sw   $ra, 0($sp)
sw   $s1, 4($sp)
sw   $a0, 8($sp)
slti $t0, $a0, 2
beq  $t0,$0,L1
addi $v0,$a0,0
j    EXIT
L1:
addi $a0,$a0, -1
jal  fib
addi $s1,$v0,0
addi $a0,$a0,-1
jal  fib
add  $v0,$v0,$s1
EXIT:
lw   $ra,0($sp)
lw   $a0,8($sp)
lw   $s1,4($sp)
addi $sp,$sp, 12
jr   $ra
```

   (a) (02%) According to the MIPS calling convention, what are the usages of \$v0 and \$a0?
   (b) (03%) In the procedure prologue, three registers were saved: \$ra, \$s1, and \$a0. \$s1 is a callee saved register, so it must be saved, but why does this procedure save registers \$ra and \$a0?
   (c) (03%) In this code, \$s1 is used as a temporary to hold fib(n-1)+fib(n-2), so why not use a temporary register, say \$t1, instead?

6. **(10%)** The overflow of 8-bit addition A+B=Sum ($A = A_7 A_6 \ldots A_0$, $B = B_7 B_6 \ldots B_0$, $S = S_7 S_6 \ldots S_0$, and carry $C_8 C_7 C_6 \ldots C_0$) can be summarized as the following conditions.

	+/-	A	B	Sum	A_7	B_7	S_7
1	A+B	+	+	—	(b)		
2	A+B	—	—	+	A_7=1, B_7=1, and S_7=0		
3	A-B	+	—	—	Same as 1		
4	A-B	—	+	+	Same as 2		

16 ÷

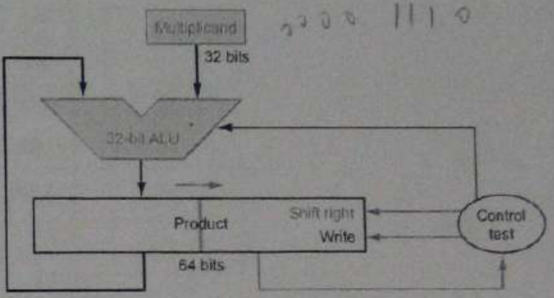
(a) Give an overflow example for an 8-bit addition in case 2.

(b) In fact, the 2nd case gives C_8=1 and C_7=0, because A_7=1, B_7=1, and S_7=0. Express the other three cases in terms of A_7, B_7, S_7, C_8, and C_7.

(c) Show the overall overflow condition in terms of C_8 and C_7.

7. **(8%)** Fill in the table for the Multiplicand and Product for each step in order to perform 2x7. You need to provide the OPERATION of the step being performed (shift right, add, no-op, sub). The value of Multiplicand is 0010 and Multiplier is initially 0111.
Bonus: (+4%) Complete the same table but using Booth Algorithm. $2 \times 7 = 14$.

Multiplicand	product	operation	steps
0010	0000 0111	Initial Values	step 0
			step 1
			step 2
			step 3
			step 4
			step 5
			step 6
			step 7
			step 8



8. **(13%)** Compiling a while loop in C or i : $s3 $t0 = r8

(a) (10%) Assume i and k correspond to $s3 (r19) and $s5 (r21), and the base of the array save is in $s6 (r22). Write the rest of other MIPS instructions corresponding for the C code: (Hint: use a *nor* instruction to perform *not*, $t0=r8,...) k : $s5 Array : $s6

```
while ( save[i] == k) {
   save[i]= ~save[i];
   i++;}
```

```
Loop:sll     $t1,$s3,2   # Temp reg $t1 = 4 * i
                         # $t1 = address of save[i]
                         # Temp reg $t0 = save[i]
                         # go to Exit if save[i] ≠ k
        ......           ......
     j   Loop            # go to Loop
Exit:
```

(b) (3%) Assume the program is started from 0x80000, give the MIPS machine code in the following table.

	opcode					function
80000	0	0	19	9	2	0
80004						
80008						
80012						
800‥	2					
800‥	…					

Hint: you may need the following information:

instruction	opcode	Funct if any	instruction	opcode	Funct if any
add	0	32	lw	35	-
sll	0	0	bne	5	-
sub	0	34	j	2	-
nor	0	27	addi	8	-

1、

R						I			
6	5	5	5	5	6	6	5	5	16
op	rs	rt	rd	sh	func	op	rs	rt	const or add

+2.5

(b) sw , store rs to rt

(a) lw , load rs to rt

(c) branch when rs (source) = rt (destinction)

(d) add i rt = rs + immidiate constant

2、

(a) because 1111 11 ... 11 1111 , all 1

+3

(b) because ABBA ABBA pattern重複,規律,且從前或後解讀一樣

(d) because 都指向 0x0000 0000

3、 +4

branch : 6 5 5 16 can branch to $PC \pm (2^{17} - 1)$

Jmp : 6 26 can jump to $PC_{31\sim28}(Addr \times 4)$

(b) because branch can go both forward and backward, and the address is withing its reachable range.

(c) because Jump can jump to $PC_{31\sim28} + (Addr \times 4)$, and $0_x00040060$ is withing its range

我字很醜>"< (但人很帥)

Yes or NO :

~~+10.5~~ +12

1) Yes, while CPU loading approaches 0, most power consumption of a computer is still, say 40% of a full working CPU.

2) No. If powerful instruction is hard to implenent in hardware, it may lower the overall performance.

3) ~~Yes~~ [illegible]. If n > 33, it may cause some issue. on the other hand If n < 0, it should be left shift instead of right shift

4) Yes, because all registers are 32 bits, so they need to extend in order to 塡滿 32 bits.

5) Yes. because the stack pointer 移動 too much during a procedure. Its hard for us to locate local variables just by stack pointer. The introduction of Frame Pointer solved the problem.

6) No, Its because the 散熱問題

PART II :

1. MIPS = $\frac{\text{Instructions}}{\text{second}} = \frac{\text{CPU Cycle}}{\boxed{\text{CPI}} \times \overset{10^6}{\text{seconds}}}$ #

5

Its $\boxed{\text{not}}$ proper to use MIPS as a performance metric. Because it depends too heavily on $\boxed{\text{CPI}}$, which is not fair.

2,

(a)

3

Ap1 by C1 : $\frac{4\times(4\times\cancel{10}^9) + 2\times(3\times\cancel{10}^{\overset{10}{\cancel{9}}}) + 2\times(2\times\cancel{10}^9)}{4\times\cancel{10}^9} = \frac{16+60+4}{4} = 20(s)$

Ap 2 by C_1 : $\frac{4\times(4\times\cancel{10}^{\overset{10}{\cancel{12}}}) + 2\times(8\times\cancel{10}^9) + 2\times(4\times\cancel{10}^9)}{4\times\cancel{10}^9} = \frac{\overset{160}{\cancel{40}}+16+8}{4} = 16\ (s)$

46

相加

Ap1 by C_2 : $\frac{4\times(2\times\cancel{10}^{\overset{10}{\cancel{10}}}) + 2\times(8\times\cancel{10}^9) + 2\times(8\times\cancel{10}^9)}{4\times\cancel{10}^9} = \frac{80+16+16}{4} = \frac{112}{4} = 28\ (s)$

Ap_2 by C_2 : $\frac{4\times(1.2\times\cancel{10}^{\overset{10}{\cancel{10}}}) + 2\times(1.6\times\cancel{10}^{\overset{10}{\cancel{10}}}) + 2\times(8\times\cancel{10}^{\overset{10}{\cancel{10}}})}{4\times\cancel{10}^9} = \frac{48+32+160}{4} = 60\ (s)$

(b) 6

machine A:

A_{P_1} by C_2 : 28(s)

A_{P_2} by C_2 : 60 (s)

workload time : $(28\times4)+60 = 112+60 = 172$ (s) #

✓

machine B:

A_{P_1} by C_2 :

$$\frac{6\times(2\times\overset{10}{\cancel{10^{10}}})+1\times(8\times\cancel{10^9})+1\times(8\times\cancel{10^9})}{8\times\cancel{10^9}} = \frac{120+8+8}{8} = \frac{136}{8} = 17\ (s)$$

A_{P_2} by C_2 :

$$\frac{6\times(1.2\times\overset{10}{\cancel{10^{10}}})+1\times(1.6\times\overset{10}{\cancel{10^{10}}})+1\times(8\times\overset{10}{\cancel{10^{10}}})}{8\times\cancel{10^9}} = \frac{72+16+80}{8} = \frac{168}{8} = 21\ (s)$$

workload time : $(17\times4)+21 = 68+21 = 89$ (s) #

✓

machine C:

A_{P_1} by C_2 :

$$\frac{5\times(2\times\overset{10}{\cancel{10^{10}}})+2\times(8\times\cancel{10^9})+1\times(8\times\cancel{10^9})}{2\times\cancel{10^9}} = \frac{100+16+8}{2} = \frac{124}{2} = 62\ (s)$$

A_{P_2} by C_2 :

$$\frac{5\times(1.2\times\overset{10}{\cancel{10^{10}}})+2\times(1.6\times\overset{10}{\cancel{10^{10}}})+1\times(8\times\overset{10}{\cancel{10^{10}}})}{2\times\cancel{10^9}} = \frac{60+32+80}{2} = \frac{172}{2} = 86\ (s)$$

workload time : $(62\times4)+86 = 248+86 = 334$ (s) # ✓

(c) 4

average CPI $= \dfrac{4\times(1.2\times\cancel{10^{10}})+2\times(1.6\times\cancel{10^{10}})+2\times(8\times\cancel{10^{10}})}{1.2\times\cancel{10^{10}}+1.6\times\cancel{10^{10}}+8\times\cancel{10^{10}}} =$

$$= \frac{4.8+3.2+16}{1.2+1.6+8} = \frac{24}{10.8} = \frac{\overset{60\ 20}{\cancel{240}}}{\underset{27\ 9}{\cancel{108}}} = \frac{20}{9}\ \text{cycles}\ \#$$ ✓

(d

6

(d)

6

		total	I/O
machine	C :	334 (s)	56 (s)
machine	B :	89 (s)	

$\cancel{62}^{2} \times \frac{\cancel{8}^{2}}{\cancel{144}_{34}} \times 4 + \cancel{86}^{2} \times \frac{\cancel{80}^{20}}{\cancel{172}_{2\cdot 43}}$

$= 16 + 40 = 56s$

Impossible ✓ because 就算 machine C 的 I/O 時間 improve 到 $\frac{1}{\infty}$, machine C 的 workload time still higher than machine B # ✓

machine C after refine I :

A_{P_1} by C_2 : $\frac{2\times \cancel{10^{9}}^{10} + 8\times \cancel{10^{9}} + 8\times \cancel{10^{9}}}{2\times \cancel{10^{9}}} = \frac{20+8+8}{2} = 18$ (s)

A_{P_2} by C_2 : $\frac{1.2\times \cancel{10^{9}}^{10} + 1.6\times \cancel{10^{9}}^{10} + 8\times \cancel{10^{9}}^{10}}{2\times \cancel{10^{9}}} = \frac{12+16+80}{2} = \frac{108}{2} = 54$ (s)

workload time : $(18\times 4) + 54 = 72 + 54 = 126$ (s) ✓ #

3.

(a)

3

case 2 : 負 + 負 = 正

example : $\underbrace{1111 \ldots 111}_{\text{all } 1 \ 負} + \underbrace{1000 \ldots 0000}_{\text{all } 0 \ 負} = \overset{正}{\underbrace{0\,1111 \ldots 1111}_{\text{all } 1}}$ #

3 (b)

A_7 B_7 S_7 C_8 C_7

Case 1 :	Case 2 :	Case 3 :	Case 4 :
A_7 : 0	題目有了	A_7 : 0	A_7 : 1
B_7 : 0		B_7 : 0	B_7 : 1
S_7 : 1		S_7 : 1	S_7 : 0
C_8 : 0		C_8 : 0	C_8 : 1
C_7 : 1		C_7 : 1	C_7 : 0

(c)

4 when $((C_7 \text{ xor } C_8) == 1$ #

4,

(a)

by using "generate" and "propagate", we can save the time of

+3

going ripple process.

~~(b)~~ ✓.

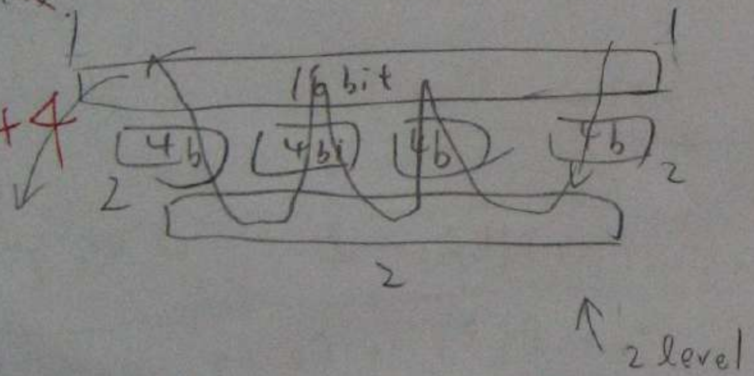
4-bit carry-look ahead adder : $1+2+1=4$ (GD)

+4.5

4-bit ripple carry adder : $1+2\times3+①=8$ (GD)

(c).

+4



A: $1+2+2+2+1 = 8$ (GD)

圖有點怪

5,

+2 (a) \$a₀ is the 參數 while \$V₀ is the return value.

(b) save \$ra : because PC 之後還要再跳回來這個 caller 的下一行

+3

save \$a₀ : because 呼叫下一個 callee 後，原本的 \$a₀ 參數也會被丟入新的值，所以要先 save 起來。

(c) because callee 沒有義務去 save #t1 , 如果 caller 自己沒有主動去把 #t1

+1 存起來的話, 可能在呼叫下一個 procedure 時 #t1 就被其他值覆蓋掉了。

7, +8

Multiplicand	product	Operation	steps
0010	0000 : 0111 0	Initial Value	0
0010	1110 : 0111 0	左減 2	1
0010	1111 : 0011 1	Shift Right	2
0010	1111 : 0011 1	NO-OP	3
0010	1111 : 1001 1	Shift Right	4
0010	1111 : 1001 1	NO-OP	5
0010	1111 : 1100 1	Shift Right	6
0010	0001 : 1100 1	左加 2	7
# 0010	0000 : 1110 0	Shift Right	8

↑

用 Booth Algorithm #

+14

(a)

```
0   Loop : sll   $t1 , $s3 , 2
4          add   $t1 , $t1 , $s6
8          lw    $t0 , 0($t1)

12         bne   $t0 , $s5 , Exit
16         nor   $t0 , $t0 , $zero        save[i] = ~save[i]
20         sw    $t0 , 0($t1)
24         addi  $s3 , $s3 , 1            i++
28         j Loop
32 Exit :
```

(b) +3.5

	OP	rs	rt	rd	sh	fun
80000	0	0	19	9	2	0
80004	0	9 ($t1)	22 ($s6)	9 ($t1)	0	32
80008	35	9	8 .		0	
80012	5	8	21		4	
80028	2	20000				
80032	Exit					