컴퓨터구조 중간고사 (2018년도 2학기)

학번: 이름:

시험시간: 10월 31일(수) 18:00-20:00

- **답이 도출되는 과정을 명확하고 알아보기 쉽게 적을 것, 답만 쓰는 경우나 풀이 과정을 알아보기 어렵게 작성한 경우 점수 없음**

---

1. (30점) 아래의 문항들에 대해 해당하는 답들을 고르시오. 풀이를 쓸 필요 없이 답만 표시하면 됨.
   [주의사항] 맞은 문항은 3점, 틀린 문항은 -1점, 답을 표시하지 않은 문항들은 0점을 부여함. 그러므로 추측으로 답을 표시하지 않기를 권장함.

   a) 5bit로 표현할 수 있는 2의 보수의 범위는?

   i. -31~+31 ii. B. -15~+15 iii. 0~+31 iv. -16~+15 v. -32~+31

   b) 4 비트로 표시된 숫자 $x = 1010_2$에 대해서 unsigned, sign and magnitude, biased notation, 1의 보수, 2의 보수의 표현 방법에 의해 나타낼 수 없는 값은?

   i. -4 ii. -6 iii. 10 iv. -2

   c) 아래의 C 코드를 수행하면 어떻게 되는가?

```
#include <stdio.h> int main() {
        int *p; *p = 5;
        printf("%d\n",*p); }
```

   i. 5를 출력 ii. garbage 값 출력 iii. 항상 비정상 종료 iv. 대부분 비정상 종료

   d) 아래의 C 코드가 수행될 때 잘못된 코드들을 모두 고르시오.

```
struct node { char *name; struct node *next; };
struct node *ar[5];
struct node **p = ar;
... /* fill ar with initialized structs */
```

   i. `&p` ii. `p->name` iii. `p[7]->next`

   iv. `*(*(p + 2))` v. `*(p[0]->next)` vi. `(*p)->next->name`

   e) 아래의 코드에 대해 틀린 설명은?

```
void funcA() {int x; printf("A");}
void funcB() {int y; printf("B"); funcA(); }
void main() {char *s = "s"; funcB();}
```

   i. x의 주소는 y보다 낮다. ii. x와 y가 속한 스택 프레임들은 주소상으로 인접해 있다.

   iii. x의 주소는 *s보다 낮다. iv. y는 스택의 시작으로부터 두번째 프레임에 위치한다.

f) MIPS 명령어 집합에 대해 다음 중 틀린 것은?

i. 조건부 분기 명령어를 이용해서 무조건 분기를 구현할 수 있다
ii. `j` 명령어만을 이용해서 loop를 구현할 수 있다. (즉 `beq`나 `bne`를 쓰지 않고)
iii. `j` 명령어를 사용하지 않고 C의 `for` loop를 구현할 수 있다.
iv. `beq` 명령어를 사용해 만들어진 모든 control flow 코드들은 `bne` 명령어를 이용해 같은 코드 라인들로 구현할 수 있다.

g) 아래의 MIP 어셈블리로 표현된 C 코드에서 괄호 부분에 들어가야 할 코드는?

```
do {i--;} while(         );
```

```
Loop:
addi $s0,$s0,-1            # i→$s0, j→$s1
slti $t0,$s1,2             # i = i - 1
beq $t0,$0, Loop           # $t0 = (j < 2)
slt $t0,$s1,$s0            # goto Loop if $t0==0
bne $t0,$0, Loop           # $t0 = (j < i) # goto Loop if $t0!=0
```

i. `j≥2&&j<i` ii. `j≥2||j<i` iii. `j<2||j≥i` iv. `j<2&&j≥i`

h) 다음 중 틀린 것은?

i. MIP는 함수를 호출하기 위해 `jal`을, 함수에서 돌아오기 위해 `jr`를 사용한다.
ii. `jal`은 `$ra` 레지스터에 PC+1 값을 저장한다.
iii. Callee 함수는 temporary register(`$ti`)을 저장/복구할 필요 없이 자유롭게 사용할 수 있다.
iv. Caller 함수는 callee 함수가 save register(`$si`) 변경하는 경우를 대비해 이를 저장/복구할 필요가 없다.

i) 두 개의 C 코드 파일들을 결합해서 하나의 실행파일을 만들 때는 코드 파일들을 각각 별도로 컴파일한 후에 하나로 합친다. 두 개 이상의 바이너리 파일(즉 오브젝트 코드)을 합칠 때 아래에서 틀린 내용과 맞는 내용을 바르게 구분한 것은?

1) jump 명령어들은 원래의 바이너리 파일에서 변화가 없다
2) Branch 명령어들은 원래의 바이너리 파일에서 변화가 없다.

i. 1) False, 2) False ii. 1) False, 2) True iii. 1) True, 2) False iv. 1) True, 2) True

j) 다음 중 맞는 것은?

i. `$rt` (target register)은 명령어의 실행 결과를 저장하는 경우가 없기 때문이 이름이 잘못 붙여진 것이다.
ii. MIPS 명령어의 모든 필드 값들은 부호 없는 양의 정수만을 가정한다.
iii. Branch(분기) 명령어로부터 $2^{16} \times 4 = 2^{18}$ 바이트만큼 떨어진 명령어로 분기 할 수 있다.
iv. 주소 상 앞으로 (즉 주소 값이 증가하는 방향) 분기하는 경우 뒤로 (주소 값이 감소) 분기하는 경우보다 더 멀리 떨어진 명령어들로 분기할 수 있다.

2. (15점) 아래의 조건에 대해서 label 분기하기 위한 MIPS 어셈블리를 각각 두 개의 MIPS 명령어들로 구현하시오.

(a) `$s0 < $s1` (b) `$s0 <= $s1` (c) `$s0 > 1`

3. (10점) MIPS to C

아래의 MIPS 코드는 일반적인 C 코드를 구현한 것이다. MIPS 코드로부터 C 코드를 유추하려고 한다 (이를 reverse engineering이라고 한다). 스택에 레지스터를 저장하는 등의 함수 호출과 관련된 부분은 고려하지 않는다. 아래와 같이 가정하자.

- `$s0` 는 C 코드의 변수 `b`를 저장하고 있음, `$s1` 는 C 코드의 변수 `i`를 저장하고 있음.
- `$s2` 는 C 코드의 상수 `10`을 저장하고 있음, `$s3`은 정수 배열 `a`를 가리키는 포인터를 저장하고 있음

어셈블리 코드는 아래와 같다. #은 주석을 나타낸다.

```
       add$s0, $zero, $zero    # b = 0;
       add  $s1, $zero, $zero  # i = 0;
       addi $s2, $zero, 10     # $s2 = const 10;
X:     slt $t0, $s1, $s2       # i < 10?
       bne $t0, $zero, Y       # branch if i < 10
       sll  $t1, $s1, 2        # $t1 = i * 4;
       add  $t2, $s3, $t1      # $t2 = &a + i * 4 ... the address of a[i]
       sw   $s1, 0($t2)        # a[i] = i;
       add  $s0, $s0, $s1      # b = b + i;
       addi $s1, $s1, 1        # i = i + 1;
       j      X                # loop back to the start
Y:                             # exit:
```

위의 코드와 동일한 C 코드를 가능한 적은 수의 라인으로 쓰시오.

4. (5 점) C Memory Management

   아래 C 코드의 문제점이 무엇인지 한 문장으로 간략하게 설명하시오.

```
int* pi = malloc(314 * sizeof(int));
if(!raspberry) pi = malloc(1 * sizeof(int));
return pi;
```

5. (10점) MIPS에서의 배열

   배열 `int arr[6] = {3, 1, 4, 1, 5, 9}`가 주어져 있고, 이 배열은 주소 `0xBFFFFF00`에서 시작한다고 가정하자. 레지스터 `$S0`는 `arr`의 주소 `0xBFFFFF00`를 저장하고 있다. 정수와 포인터 변수들은 4-byte 크기라고 가정한다. 아래의 어셈블리 코드들은 어떤 동작을 하는지 한두 문장으로 간략하게 설명하시오.

   a)

```
lw  t0, 0(s0)    # Loads arr[0] into register t0
lw  t1, 8(s0)    # Loads arr[2] into register t1
add t2, t0, t1   # Sets t2 equal to t0 plus t1
sw  t2, 4(s0)    # Sets arr[1] equal to value in t2
```

   b)

```
        add  t0, x0, x0     # Sets register t0 to 0
 loop:  slti t1, t0, 6      # Sets t1 to 1 if t0 < 6, 0 otherwise
        beq  t1, x0, end    # Branches to the end if t1 is 1 (t0 >= 6)
        slli t2, t0, 2      # Sets t2 to t0 * 4 (4 is number of bytes in an integer)
        add  t3, s0, t2     # Sets t3 to the address of arr[t0] (added t2 bytes to arr)
        lw   t4, 0(t3)      # Load arr[t0] into register t4
        sub  t4, x0, t4     # Sets t4 to its negative
        sw   t4, 0(t3)      # Stores this updated value back at arr[t0]
        addi t0, t0, 1      # Increments t0 to move to the next element
        jal  x0, loop       # Jump back to the loop label
  end:
```

6. (15점) C code translation

C 코드 prog.c를 컴파일하여 얻은 MIPS 어셈블리에서 pseudo 명령어를 제거한 과정이 아래와 같다.

Step 1: 입력 C 코드	Step 2: MIPS Assembly	Step 3: True Assembly

`#include <stdio.h>`
`int main (int argc, char *argv[])`
`{`
`  int i, sum = 0;`
`  for(i=0; i<=100; i++)`
`  sum = sum + i * i;`
`  printf(`
`  "The sum of sq from 0 .. 100 is %d\n", sum);`
`}`
	`_____.text`
	`  .align 2`
	`  .globl main`
	`main:`
	`  subu  $sp, $sp, 32`
	`  …`
	`  sw  $0, 28($sp)`
	`loop:`
	`  …`
	`  sw  $t0, 28($sp)`
	`  ble  $t0,100, loop`
	`  la  $a0, str`
	`  lw  $a1, 24($sp)`
	`  jal  printf`
	`  move $v0, $0`
	`  lw  $ra, 20($sp)`
	`  addiu $sp,$sp,32`
	`  jr  $ra`
	`  .data`
	`  .align 0`
	`str:`
	`  .asciiz  "The sum of sq from 0 .. 100 is %d\n"`
		`00 addiu $29,$29,-32`
		`04 sw  $31,20($29)`
		`08 sw  $4, 32($29)`
		`0c sw  $5, 36($29)`
		`10 sw  $0, 24($29)`
		`14 sw  $0, 28($29)`
		`18 lw  $14,28($29)`
		`1c multu $14,$14`
		`20 mflo  $15`
		`24 lw  $24,24($29)`
		`28 addu  $25,$24,$15`
		`2c sw  $25,24($29)`
		`30 addiu $8,$14,1`
		`34 sw  $8,28($29)`
		`38 slti  $1,$8, 101`
		`3c bne  $1,$0, loop`
		`40 lui  $4,l.str`
		`44 ori  $4,$4, r.str`
		`48 lw  $5,24($29)`
		`4c jal  printf`
		`50 add  $2,$0, $0`
		`54 lw  $31,20($29)`
		`58 addiu $29,$29,32`
		`5c jr  $31`

위의 어셈블리 코드에 해당하는 symbol table과 relocation table은 아래와 같다.

Symbol table			Relocation Table		
Label	Address (in module)	Type	Address	Instr. type	Dependency
main:	0x00000000	global text	0x00000040	lui	l.str
loop:	0x00000018	local text	0x00000044	ori	r.str
str:	0x00000000	local data	0x0000004c	jal	printf

a) Step3의 코드에서 PC-relative 주소를 사용하는 명령어(들)을 찾아서 label를 실제 값으로 바꾸시오.

b) step 3 의 코드를 바이너리로 변환하여 prog.o 을 얻었고 Link 단계에서 이를 printf 함수가 포함된 libc.o 와 합쳐서 실행파일을 만들려고 한다. Link 과정 후의 최종 symbol table 와 relocation table 은 다음과 같다.

Symbol table			Relocation Table		
Label	Address (in module)	Type	Address	Instr. type	Dependency
main:	0x00000000	global text	0x00000040	lui	l.str
loop:	0x00000018	local text	0x00000044	ori	r.str
str:	0x10000430	local data	0x0000004c	jal	printf
printf:	0x00000cb0 ….				

Step 3 의 코드에서 Link 단계에서 결정되는 절대주소가 필요한 label 들을 사용하는 명령어(들)을 찾아서 label 들을 실제 주소 값으로 변경하시오. (주소값은 10 진수로 쓸 것)

7. (15점) Instruction Set Architecture

MIPS 명령어는 8-bit 데이터, 즉 바이트 단위의 데이터들을 다룬다. 새로운 명령어 집합(ISA)을 만들려고 하고 NIPS라고 부르기로 한다. NIPS는 8-bit 대신 4-bit 데이터, 즉 nibble 단위 데이터들을 다루는 명령어 집합으로 만들려고 한다. NIPS에서 워드의 크기는 6-nibble, 즉 24-bit라고 하자. 그리고 레지스터들도 6-nibble 크기이며, 명령어들도 6-nibble 크기로 구성된다. 메모리는 nibble 주소 단위로 접근한다. 새로운 nibble 주소 체계에서 명령어들과 데이터 워드들의 시작 주소는 6의 배수로 정렬되어 있다. 즉 워드 0은 nibble 0, 워드 1은 nibble 6, 워드 2는 nibble 12 등의 주소에서 시작하게 된다. **NIPS는 총 10개의 24-bit 레지스터를 사용한다.**

32-bit MIPS ISA의 명령어 형식과 필드들을 NIPS를 설계하는데 적용하려고 한다. NIPS의 명령어들과 워드 데이터들이 24-bit 크기이고 메모리 주소가 nibble 단위로 관리되는 것을 빼면 많은 부분들이 MIPS와 유사하다.

(a) 아래의 그림에 24-bit ISA인 NIPS의 R-type와 I-type의 명령어 형식의 필드들의 크기를 bit 단위로 채우시오.

opcode	rs	rt	rd	shamt	funct
6					

opcode	rs	rt	imm
6			

(b) `rs`, `rt`, `rd` 필드의 크기를 고려할 때 NIPS에서 사용할 수 있는 레지스터의 최대 갯수는?

(c) NIPS에서 주어진 opcode에서 사용 가능한 function의 최대 갯수는?

(d) NIPS에서 PC에 **nibble 주소**로 1566 가 저장되어 있다면, branch 명령어를 통해 점프할 수 있는 가장 큰 nibble 주소는 십진수로 얼마인가?

(e) NIPS 형식으로 인코딩된 아래의 24-bit 명령어를 32-bit MIPS 어셈블리로 변환하여 2진수로 나타내고 각 MIPS 명령어의 각 필드들을 표시하시오.

0x8C2408

변환 시 레지스터 표기는 `$0`, `$1`,... 등과 같은 레지스터 번호를 사용하고 MIPS ISA와 같은 opcode를 사용한다고 가정하시오.